用 Python 分析数据、
预测结果的简单高效的方式

WILEY 异步图书 www.epubit.com.cn

Python 机器学习

预测分析核心算法

[美] Michael Bowles 著
沙赢 李鹏 译

MACHINE LEARNING IN PYTHON

ESSENTIAL TECHNIQUES
FOR PREDICTIVE ANALYSIS

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

WILEY

Python 机器学习

预测分析核心算法

[美] Michael Bowles 著

沙赢 李鹏 译

MACHINE LEARNING IN PYTHON

ESSENTIAL TECHNIQUES FOR PREDICTIVE ANALYSIS

人 民 邮 电 出 版 社

北 京

异步社区电子书

感谢您购买异步社区电子书！异步社区已上架电子书 500 余种，社区还会经常发布福利信息，对社区有贡献的读者赠送免费样书券、优惠码、积分等等，希望您在阅读过程中，把您的阅读体验传递给我们，让我们了解读者心声，有问题我们会及时修正。

社区网址： http://www.epubit.com.cn/

反馈邮箱： contact@epubit.com.cn

异步社区里有什么？

图书、电子书（半价电子书）、优秀作译者、访谈、技术会议播报、赠书活动、下载资源。

异步社区特色：

纸书、电子书同步上架、纸电捆绑超值优惠购买。

最新精品技术图书全网首发预售。

晒单有意外惊喜！

异步社区里可以做什么？

博客式写作发表文章，提交勘误赚取积分，积分兑换样书，写书评赢样书券等。

联系我们：

微博：

@ 人邮异步社区　　@ 人民邮电出版社 - 信息技术分社

微信公众号：

人邮 IT 书坊　　异步社区

QQ 群： 368449889

图书在版编目（ＣＩＰ）数据

Python机器学习 ： 预测分析核心算法 / （美） 鲍尔斯 (Michael Bowles) 著 ; 沙嬴, 李鹏译. -- 北京 : 人民邮电出版社, 2017.1
ISBN 978-7-115-43373-2

Ⅰ. ①P… Ⅱ. ①鲍… ②沙… ③李… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.56

中国版本图书馆CIP数据核字(2016)第245035号

版权声明

◆ 著　[美] Michael Bowles
译　沙　嬴　李　鹏
责任编辑　陈冀康
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：21
字数：445 千字　2017 年 1 月第 1 版
印数：1 – 3 000 册　2017 年 1 月北京第 1 次印刷

著作权合同登记号　图字：01-2015-4678 号

定价：69.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

内容提要

在学习和研究机器学习的时候，面临令人眼花缭乱的算法，机器学习新手往往会不知所措。本书从算法和 Python 语言实现的角度，帮助读者认识机器学习。

本书专注于两类核心的“算法族”，即惩罚线性回归和集成方法，并通过代码实例来展示所讨论的算法的使用原则。全书共分为 7 章，详细讨论了预测模型的两类核心算法、预测模型的构建、惩罚线性回归和集成方法的具体应用和实现。

本书主要针对想提高机器学习技能的 Python 开发人员，帮助他们解决某一特定的项目或是提升相关的技能。

致我的孩子——Scott、Seth 和 Cayley，他们那如花的生命带给我世界上最美妙的时光。

致我亲密的朋友——David 和 Ron，他们给我带来了无私慷慨、坚定不移的友谊。

致我在加州山景城黑客道场的朋友和同事，他们给我带来了技术挑战以及机敏的应答。

致我的攀岩伙伴。正如一个伙伴凯瑟琳所说通过攀岩可以交上最好的朋友，因为“他们见证过因恐惧而僵硬的面庞，见证过相互鼓励搀扶，见证过攀登成功后的喜悦”。

作者简介

Michael Bowles 拥有机械工程学士和硕士学位、仪器仪表博士学位以及 MBA 学位。他的履历涉及学术界、工业界以及商业界。他目前在一家初创公司工作，其中机器学习技术至关重要。他是多个管理团队的成员、咨询师以及顾问。他也曾经在加州山景城的黑客道场、创业公司孵化器和办公场所教授机器学习课程。

他出生于俄克拉荷马州并在那里获得学士和硕士学位。在东南亚待了一段时间后，他前往剑桥攻读博士学位，毕业后任职于 MIT 的 Charles Stark Draper 实验室[①]。之后他离开波士顿前往南加州的休斯飞机公司开发通信卫星。在 UCLA 获得 MBA 学位后，他前往旧金山的湾区工作。作为创始人以及 CEO，他目前经营两家公司，这两家公司都已获风险投资。

他目前仍然积极参与技术以及创业相关的工作。近期项目包括使用机器学习技术进行自动交易，基于基因信息进行生物预测，使用自然语言处理技术进行网站优化，利用人口统计学及实验室数据预测医疗效果，在机器学习和大数据相关领域的公司里尽心尽责。可以通过 www.mbowles.com 联系到他。

① Charles Stark Draper 是一名美国科学家和工程师，被称为“惯性导航之父”。他是 MIT 仪器实验室的创始人，后来此实验室用其名来命名，此实验室设计了阿波罗登月计划中的阿波罗导航计算机。

技术编辑简介

Daniel Posner 拥有经济学学士以及硕士学位，在波士顿大学完成生物统计学博士学位。他曾为医药生物领域的公司以及帕罗奥图退伍军人事务部医院的研究人员提供统计学方面的咨询。

Daniel 和本书作者就书中相关主题有过广泛的合作。他们曾经一起撰写过开发 Web 级梯度提升算法的项目申请书。最近，他们合作利用随机森林和样条基扩展技术解决药物研制过程中的关键变量识别问题，其目标是提升预测效果以减少试验所需样本规模。

致谢

首先感谢 Wiley 出版社工作人员在本书创作期间提供的大量帮助。最早是组稿编辑 Robert Elliot 和我联系写作本书，他很容易相处。之后是 Jennifer Lynn 为本书的编辑。她对每个问题都积极响应，写作期间非常耐心地和我联系，保证我的写作计划如期完成。非常感谢你们的工作。

我也非常感谢如此敏锐、缜密的统计学家兼程序员 Daniel Posner 作为本书的技术编辑，从他那获得了巨大的安慰和帮助。感谢你的工作，也感谢你在机器学习、统计学以及算法上所做的有趣讨论。我还没见过其他谁的思维可以达到如此深入、如此迅捷。

前言

从数据中提取有助于决策的信息正在改变着现代商业的组织，同时也对软件开发人员产生了直接的影响。一方面是对新的软件开发技能的需求，市场分析师预计到 2018 年对具有高级统计和机器学习技术的人才需求缺口将达 140000 ～ 190000 人。这对具有上述技能的人员来说意味着丰厚的薪水和可供选择的多种有趣的项目。另一方面对开发人员的影响就是逐步出现了统计和机器学习相关的核心工具，这减轻了开发人员的负担。当他们尝试新的算法时，不需要重复发明“轮子”。在所有通用计算机语言中，Python 的开发人员已经站在了最前沿，他们已开发了当前最先进的机器学习工具包，但从拥有这些工具包到如何有效地使用它们仍然存在一定的距离。

开发人员可以通过在线课程、阅读质量上乘的书籍等方式来获得机器学习的相关知识。它们通常都对机器学习的算法、应用实例给出了精彩的阐述，但是因为当前算法如此之多，以至于很难在一门课程或一本书中覆盖全部算法的相关细节。

这给实践者带来了困难。当面临众多算法时，机器学习新手可能需要多次尝试才能做出决定，这往往需要开发人员来填补从问题提炼到最终问题解决之间的所有算法使用方便的细节。

本书尝试填补这一鸿沟，所采用的方法就是只集中于两类核心的“算法族”，这两类算法族已在广泛的应用领域中证明了其最佳的性能。此论断的证据如下：这两类算法在众多机器学习算法竞争者中已获得支配性地位，新开发的机器学习工具包都会率先支持此两类算法，以及研究工作给出的性能对比结论（见第 1 章）。重点关注这两类算法使我们可以更详细地介绍算法的使用原则，并通过一系列的示例细节来展示针对不同问题如何使用这些算法。

本书主要通过代码实例来展示所讨论的算法的使用原则。以我在加州山景城的黑客道场（Hacker Dojo）授课的经验来看，我发现开发人员更愿意通过直接看代码示例来了解算法原理，而不是通过数学公式推导。

本书使用 Python 语言，因为它能提供将功能和专业性良好结合的机器学习算法包。Python 是一种经常使用的计算机语言，以产生精炼、可读性代码而著称。这导致目前已有相当数量的业界旗舰公司采用 Python 语言进行原型系统开发和部署。Python 语言开发人员获得了广泛的支持，包括大量的同业开发人员组成的社区、各种开发工具、扩展库等等。Python 广泛应用于企业级应用和科学研究领域。它有相当数量的工具包支持计算密集型应用，如机器学习。它也收集了当前机器学习领域的代表性算法（这样就可以省去重复性劳动）。Python 相比专门的统计语言如 R 或 SAS（Statistical Analysis System）是一门更通用的语言，它的机器学习算法包吸收了当前一流的算法，并且在一直扩充。

本书的目标读者

本书主要面向想提高机器学习技能的 Python 开发人员，不管是针对某一特定的项目，还是只想提升相关技能。开发人员很可能在工作中遇到新问题需要使用机器学习的方法来解决。当今机器学习的应用领域如此之广，使其已成为简历中一项十分有用的技能。

本书为 Python 开发人员提供如下内容：

- 机器学习所解决的基本问题的描述；
- 当前几种最先进的算法；
- 这些算法的应用原则；
- 一个机器学习系统的选型、设计和评估的流程；
- 流程、算法的示例；
- 可进一步修改扩展的代码。

为了能够顺利地理解这本书，读者所需的背景知识包括：了解编程、能够读写代码。因为本书的代码示例、库、包都是 Python 语言的，所以本书主要适用于 Python 开发人员。本书通过运行算法的核心代码来展示算法的使用原则，然后使用含有此算法的工具包来展示如何应用此算法来解决问题。开发人员通过源代码可以获得对算法的直观感受，就像其他人通过数学公式的推导来掌握算法。一旦掌握了算法的核心原理，应用示例就直接使用 Python 工具包，这些工具包都包含了能够有效使用这些算法必需的辅助模块（如错误检测、输入输出处理、模型所需数据结构的处理、基于训练模型的预测方法的处理，等等）。

除了编程背景，懂得相关数学、统计的知识将有助于掌握本书的内容。相关数学知识包括大学本科水平的微分学（知道如何求导，少量线性代数知识）、矩阵符号的意义、矩阵乘、求逆矩阵。这些知识主要是帮助理解一些算法中的求导部分，很多情况下就是一个简单函数的求导或基本的矩阵操作。能够理解概念层面上的数学计算将有助于对算法的理解。明白推导各步的由来有助于理解算法的强项和弱项，也帮助读者面对具体的问题时，决定哪个算法是最佳选择。

本书也用到了一些概率和统计知识。对这方面的要求包括熟悉大学本科水平的概率知识和概念，如实数序列的均值、方差和相关系数。当然即使这些知识对读者来说有些陌生，也不会影响读者对代码的理解。

本书涵盖了机器学习算法的两大类：惩罚线性回归（penalized linear regression），如岭回归算法（ridge）、Lasso 算法；集成方法（ensemble methods），如随机森林算法（random forests）、梯度提升算法（gradient boosting）。上述两大类算法有一些变体，都可以解决回归和分类的问题（在本书开始部分将会介绍分类和回归的区别）。

如果读者已熟悉机器学习并只对其中的一类算法感兴趣，可以直接跳到相关的二章。每类算法由两章组成，一章介绍基本原理，另外一章介绍针对不同类型问题的用法。惩罚线性回归由下列两章组成：第 4 章“惩罚线性回归模型”和第 5 章“利用惩罚线性回归方法来构建预测模型”；集成方法由下列两章组成：第 6 章“集成方法”和第 7 章“用 Python 构建集成模型”。快速浏览第 2 章“通过理解数据来了解问题”将有助于理解算法应用章节中的问题。刚刚进入机器学习领域准备从头到尾通读的读者可以把第 2 章留到阅读那些算法应用章节前。

本书包含的内容

如上所述，本书涵盖两大类算法，这些算法近期都获得了发展，并将仍然获得持续性研究，它们都起源于早期的技术，但已使这些早期技术黯然失色。

惩罚线性回归代表了对最小二乘法回归方法（least squares regression）的相对较新的改善和提高。惩罚线性回归具有的几个特征使其成为预测分析的首选。惩罚线性回归引入了一个可调参数，使最终的模型在过拟合与欠拟合之间达到了平衡。它还提供不同的输入特征对预测结果的相对贡献的信息。上述这些特征对于构建预测模型都是十分重要的。而且，对于某些问题惩罚线性回归可以产生最佳的预测性能，特别是对于欠定的问题以及具有很多输入参数的问题，如基因领域、文本挖掘等。进一步，坐标下降法（coordinate descent methods）等新方法可以使惩罚线性回归模型训练过程运行得更快。

为了帮助读者更好地理解惩罚线性回归，本书也概要介绍了线性回归及其扩展，如逐步回归（stepwise regression），主要是希望能够培养读者对算法的直观感受。

集成方法是目前最有力的预测分析工具之一。它可以对特别复杂的行为进行建模，特别是过定的问题，通常这些都是与互联网有关的预测问题（如返回搜索结果和预测点击率）。由于集成方法的性能，许多经验丰富的数据科学家在做第一次尝试时都使用该方法。集成方法使用相对简单，而且可以依据对预测的贡献程度对输入特征排序。

目前集成方法与惩罚线性回归齐头并进。然而惩罚线性回归是从克服一般回归方法的局限性进化而来的，集成方法是从克服二元决策树的局限性进化而来的。因此本书介绍集成方法时，也会涉及二元决策树的背景知识，因为集成方法继承了二元决策树的一些属性。了解这些将有助于培养对集成方法的直觉。

本书的组织

本书遵循了着手解决一个预测问题的基本流程。开始阶段包括对数据的理解、如何形式化表示问题，然后开始尝试使用算法解决问题，评估其性能。在这个过程中，本书将

概要描述每一步采用的方法及其原因。第 1 章给出本书涵盖的问题和所用方法的完整描述，本书使用来自 UC Irvine 数据仓库的数据集作为例子；第 2 章展示了一些数据分析的方法和工具，帮助读者对新数据集具有一定的洞察力。第 3 章“预测模型的构建：平衡性能、复杂性以及大数据”主要介绍由上述三者带给预测分析技术的困难以及所采用的技术，勾勒了问题复杂度、模型复杂度、数据规模和预测性能之间的关系，讨论了过拟合问题以及如何可靠地感知到过拟合，以及不同类型问题下的性能评价标准。第 4 章、第 5 章分别介绍惩罚线性回归的背景及其应用，即如何解决第 2 章所述的问题。第 6 章、第 7 章分别介绍集成方法的背景及其应用。

如何使用本书

为了运行书中的代码示例，需要有 Python2.x、SciPy、NumPy、Pandas 和 scikit-learn。由于交叉依赖和版本的问题，这些软件的安装可以会有些困难。为了简化上述软件安装过程，可以使用来自 Continuum Analytics（http://continuum.io/）的这些包的免费分发版。Continuum Analytics 提供的 Anaconda 软件包可自由下载并且包含 Python2.x 在内的运行本书代码所需的全部软件包。我在 Ubuntu 14.04 Linux 发行版上测试了本书的代码，但是没有在其他的操作系统上测试过。

约定

为了便于对本书的理解，本书通篇采用如下的约定。

警告 这些方框表示是与其周围文本直接相关的不能忘记的重要信息。

注释 表示与当前讨论相关的注释、说明、提示和技巧。

源代码

研究本书示例代码时，可以选择手工敲入这些代码，也可以直接使用随书带的源代码文件。本书用到的所有源代码都可以从 http://www.wiley.com/go/pythonmachinelearning 中下载获得。在本书代码片段旁会有一个下载的小图标，并注明文件名，这样就可以知道此文件在下载的源代码中，并且可以很轻松地从下载源代码中找到此文件。读者可访问上述网站，定位到本书书名（可以使用搜索框或书名列表），在本书详细介绍页面点击“Download Code”链接就可以获得本书的全部源代码。

注释 因为很多书的书名都非常相似，最简单的方法就是通过 ISBN 来查找，本书的 ISBN 是 978-1-118-96174-2。

下载源代码后，只需用你惯用的解压缩工具解压缩即可。

勘误表

我们已尽最大可能避免在文本和代码中出现错误。但是没有人是完美的，同样错误也难免会发生。如果读者在书中发现了错误，诸如拼写错误、代码错误等等，能及时反馈，我们将非常感谢。提交勘误表，能减少其他读者的困惑，同时也帮助我们提供更高质量的内容。

获得本书勘误表的方法：访问 http://www.wiley.com，通过搜索框或书名列表定位到本书；然后进入本书的详细介绍页面，点击“Book Errata”链接，可以看到关于本书所有已提交的并由 Wiley 出版社编辑上传的勘误表。

目录

第 1 章　关于预测的两类核心算法……1

1.1　为什么这两类算法如此有用……1

1.2　什么是惩罚回归方法……6

1.3　什么是集成方法……8

1.4　算法的选择……9

1.5　构建预测模型的流程……11

1.5.1　构造一个机器学习问题……12

1.5.2　特征提取和特征工程……14

1.5.3　确定训练后模型的性能……15

1.6　各章内容及其依赖关系……15

小结……17

参考文献……17

第 2 章　通过理解数据来了解问题……19

2.1　“解剖”一个新问题……19

2.1.1　属性和标签的不同类型决定模型的选择……21

2.1.2　新数据集的注意事项……22

2.2　分类问题：用声纳发现未爆炸的水雷……23

2.2.1　“岩石 vs. 水雷”数据集的物理特性……23

2.2.2　“岩石 vs. 水雷”数据集统计特征……27

2.2.3　用分位数图展示异常点……30

2.2.4　类别属性的统计特征……32

2.2.5　利用 Python Pandas 对“岩石 vs. 水雷”数据集进行统计分析……32

2.3　对“岩石 vs. 水雷”数据集属性的可视化展示……35

2.3.1　利用平行坐标图进行可视化展示……35

2.3.2　属性和标签的关系可视化……37

2.3.3　用热图（heat map）展示属性和标签的相关性……44

2.3.4　对“岩石 vs. 水雷”数据集探究过程小结……45

2.4　基于因素变量的实数值预测：鲍鱼的年龄……45

2.4.1　回归问题的平行坐标图：鲍鱼问题的变量关系可视化……51

2.4.2　回归问题如何使用关联热图——鲍鱼问题的属性对关系的可视化……55

2.5　用实数值属性预测实数值目标：评估红酒口感……57

2.6　多类别分类问题：它属于哪种玻璃……63

小结……68

参考文献……69

第 3 章　预测模型的构建：平衡性能、复杂性以及大数据……71

3.1　基本问题：理解函数逼近……71

3.1.1　使用训练数据……72

3.1.2　评估预测模型的性能……73

3.2　影响算法选择及性能的因素——复杂度以及数据……74

3.2.1　简单问题和复杂问题的对比……74

3.2.2　一个简单模型与复杂模型的对比……77

3.2.3　影响预测算法性能的因素……80

3.2.4　选择一个算法：线性或者

非线性……81
3.3 度量预测模型性能……81
3.3.1 不同类型问题的性能评价指标……82
3.3.2 部署模型的性能模拟……92
3.4 模型与数据的均衡……94
3.4.1 通过权衡问题复杂度、模型复杂度以及数据集规模来选择模型……94
3.4.2 使用前向逐步回归来控制过拟合……95
3.4.3 评估并理解你的预测模型……101
3.4.4 通过惩罚回归系数来控制过拟合——岭回归……103
小结……112
参考文献……112

第 4 章 惩罚线性回归模型……113
4.1 为什么惩罚线性回归方法如此有效……113
4.1.1 足够快速地估计系数……114
4.1.2 变量的重要性信息……114
4.1.3 部署时的预测足够快速……114
4.1.4 性能可靠……114
4.1.5 稀疏解……115
4.1.6 问题本身可能需要线性模型……115
4.1.7 什么时候使用集成方法……115
4.2 惩罚线性回归：对线性回归进行正则化以获得最优性能……115
4.2.1 训练线性模型：最小化错误以及更多……117
4.2.2 向 OLS 公式中添加一个系数惩罚项……118
4.2.3 其他有用的系数惩罚项：Manhattan 以及 ElasticNet……118
4.2.4 为什么套索惩罚会导致稀疏的系数向量……119
4.2.5 ElasticNet 惩罚项包含套索惩罚项以及岭惩罚项……120
4.3 求解惩罚线性回归问题……121
4.3.1 理解最小角度回归与前向逐步回归的关系……121
4.3.2 LARS 如何生成数百个不同复杂度的模型……125
4.3.3 从数百个 LARS 生成结果中选择最佳模型……127
4.3.4 使用 Glmnet ：非常快速并且通用……133
4.4 输入为数值型数据的线性回归方法的扩展……140
4.4.1 使用惩罚回归求解分类问题……140
4.4.2 求解超过 2 种输出的分类问题……145
4.4.3 理解基扩展：使用线性方法来解决非线性问题……145
4.4.4 向线性方法中引入非数值属性……148
小结……152
参考文献……153

第 5 章 使用惩罚线性方法来构建预测模型……155
5.1 惩罚线性回归的 Python 包……155
5.2 多变量回归：预测红酒口感……156
5.2.1 构建并测试模型以预测红酒口感……157
5.2.2 部署前在整个数据集上进行训练……162
5.2.3 基扩展：基于原始属性扩展新属性来改进性能……168
5.3 二分类：使用惩罚线性回归来检测未爆炸的水雷……172
构建部署用的岩石水雷分类器……183

5.4 多类别分类 - 分类犯罪现场的玻璃样本 ······ 196
小结 ······ 201
参考文献 ······ 202

第 6 章 集成方法 ······ 203
6.1 二元决策树 ······ 204
6.1.1 如何利用二元决策树进行预测 ······ 205
6.1.2 如何训练一个二元决策树 ······ 207
6.1.3 决策树的训练等同于分割点的选择 ······ 211
6.1.4 二元决策树的过拟合 ······ 214
6.1.5 针对分类问题和类别特征所做的修改 ······ 218
6.2 自举集成：Bagging 算法 ······ 219
6.2.1 Bagging 算法是如何工作的 ······ 219
6.2.2 Bagging 算法小结 ······ 230
6.3 梯度提升法（Gradient Boosting） ······ 230
6.3.1 梯度提升法的基本原理 ······ 230
6.3.2 获取梯度提升法的最佳性能 ······ 234
6.3.3 针对多变量问题的梯度提升法 ······ 237
6.3.4 梯度提升方法的小结 ······ 241
6.4 随机森林 ······ 241
6.4.1 随机森林：Bagging 加上随机选择的属性子集 ······ 246
6.4.2 随机森林的性能 ······ 246
6.4.3 随机森林小结 ······ 247
小结 ······ 248
参考文献 ······ 248

第 7 章 用 Python 构建集成模型 ······ 251
7.1 用 Python 集成方法工具包解决回归问题 ······ 251
7.1.1 构建随机森林模型来预测红酒口感 ······ 251
7.1.2 用梯度提升法预测红酒品质 ······ 258
7.2 用 Bagging 来预测红酒口感 ······ 266
7.3 Python 集成方法引入非数值属性 ······ 271
7.3.1 对鲍鱼性别属性编码引入 Python 随机森林回归方法 ······ 271
7.3.2 评估性能以及变量编码的重要性 ······ 274
7.3.3 在梯度提升回归方法中引入鲍鱼性别属性 ······ 276
7.3.4 梯度提升法的性能评价以及变量编码的重要性 ······ 279
7.4 用 Python 集成方法解决二分类问题 ······ 282
7.4.1 用 Python 随机森林方法探测未爆炸的水雷 ······ 282
7.4.2 构建随机森林模型探测未爆炸水雷 ······ 283
7.4.3 随机森林分类器的性能 ······ 288
7.4.4 用 Python 梯度提升法探测未爆炸水雷 ······ 289
7.4.5 梯度提升法分类器的性能 ······ 296
7.5 用 Python 集成方法解决多类别分类问题 ······ 300
7.5.1 用随机森林对玻璃进行分类 ······ 300
7.5.2 处理类不均衡问题 ······ 304
7.5.3 用梯度提升法对玻璃进行分类 ······ 306
7.5.4 评估在梯度提升法中使用随机森林基学习器的好处 ······ 311
7.6 算法比较 ······ 313
小结 ······ 315
参考文献 ······ 315

第 1 章

关于预测的两类核心算法

本书集中于机器学习领域，只关注那些最有效和获得广泛使用的算法。不会提供关于机器学习技术领域的全面综述。这种全面性的综述往往会提供太多的算法，但是这些算法并没有在从业者中获得积极的应用。

本书涉及的机器学习问题通常是指“函数逼近（function approximation）”问题。函数逼近问题是有监督学习（supervised learning）问题的一个子集。线性回归和逻辑回归是解决此类函数逼近问题最常见的算法。函数逼近问题包含了各种领域中的分类问题和回归问题，如文本分类、搜索响应、广告放置、垃圾邮件过滤、用户行为预测、诊断等。这个列表几乎可以一直列下去。

从广义上说，本书涵盖了解决函数逼近问题的两类算法：惩罚线性回归和集成方法。本章将介绍这些算法，概述它们的特性，回顾算法性能对比研究的结果，以证明这些算法始终如一的高性能。

然后本章讨论了构建预测模型的过程，描述了这里介绍的算法可以解决的问题类型，以及如何灵活地构建问题模型，选择用于做预测的特征。本章也描述了应用算法的具体步骤，包括预测模型的构建、面向部署的性能评估等。

1.1　为什么这两类算法如此有用

有几个因素造就了惩罚线性回归和集成方法成为有用的算法集。简单地说，面对实践中遇到的绝大多数预测分析（函数逼近）问题，这两类算法都具有最优或接近最优的性能。这些问题包含：大数据集、小数据集、宽数据集（wide data sets）[①]、高瘦数据集（tall skinny data sets）[②]、复杂问题、简单问题，等等。Rich Caruana 及其同事的两篇论文为上述论断提供了证据。

1．“An Empirical Comparison of Supervised Learning Algorithms,” Rich Caruana，Alexandru Niculescu-Mizi。

[①] 宽数据集（wide data set）指每次观测时有大量的测量项，但是观测次数有限的数据。若把数据看成表格形式，则此类数据集列数很多，而行数有限。典型的此类数据集包括神经影像、基因组以及其他生物医学方面的。——译者注。

[②] 高瘦数据集（tall skinny data set）指每次观测时测量项有限，但是进行了大量的观测。若把数据看成表格的形式，则此类数据集列数有限，行数很多。典型的此类数据集包括临床试验数据、社交网络数据等。——译者注。

2．“An Empirical Evaluation of Supervised Learning in High Dimensions,” Rich Caruana, Nikos Karampatziakis 和 Ainur Yessenalina。

在这两篇论文中，作者选择了各种分类问题，用各种不同的算法来构建预测模型。然后测试这些预测模型在测试数据中的效果，这些测试数据当然不能应用于模型的训练阶段，对这些算法根据性能进行打分。第一篇论文针对 11 个不同的机器学习问题（二元分类问题）对比了 9 个基本算法。所选问题来源广泛，包括人口统计学、文本处理、模式识别、物理学和生物学。表 1-1 列出了此篇论文所用的数据集，所用名字与论文中的一致。此表还展示了针对每个数据集做预测时使用了多少属性（特征）以及正例所占的百分比。

术语“正例”（positive example）在分类问题中是指一个实验（输入数据集中的一行数据）其输出结果是正向的（positive）。例如，如果设计的分类器是判断雷达返回信号是否表明出现了一架飞机，那么正例则是指在雷达视野内确实有一架飞机的那些结果。正例这个词来源于这样的例子：两个输出结果分别代表出现或不出现。其他例子还包括在医学检测中某种疾病出现或不出现，退税中是否存在欺骗。

不是所有的分类问题都处理出现或不出现的问题。例如通过计算机分析一个作家发表的作品或者其手写体的样本来判断此人的性别：男性或女性，在这里性别的出现或不出现是没有什么意义的。在这些情况下，指定哪些为正例、哪些为负例则有些随意，但是一旦选定，在使用中要保持一致。

在第 1 篇论文的某些数据集中，某一类的数据（例子）要远多于其他类的数据（例子），这叫作非平衡（unbalanced）。例如，2 个数据集 Letter.p1 和 Letter.p2. 都是用于解决相似的问题：在多种字体下正确地分出大写字母。Letter.p1 的任务是在标准字母的混合集中正确区分出大写字母 O，Letter.p2 的任务是将字母正确划分成 A-M 和 N-2 的两类。表 1-1 中的“正例百分比”一栏反映了这种数据非平衡的差异性。

表 1-1 机器学习算法比较研究中问题的梗概

数据集名称	属　性　数	正例百分比
Adult	14	25
Bact	11	69
Cod	15	50
Calhous	9	52
Cov_Type	54	36
HS	200	24
Letter.p1	16	3
Letter.p2	16	53

续表

数据集名称	属　性　数	正例百分比
Medis	63	11
Mg	124	17
Slac	59	50

表 1-1 还显示了每个数据集所使用的属性（特征）的数量。特征就是基于此进行预测的变量。例如，预测一架飞机能否按时到达目的地，可能导入下列属性（特征）：航空公司的名字、飞机的制造商和制造年份、目的地机场的降水量和航线上的风速与风向等等。基于很多特征做预测很可能是一件很难说清楚是福还是祸的事情。如果导入的特征与预测结果直接相关，那么当然是一件值得祝福的事情。但是如果导入的特征与预测结果无关，就是一件该诅咒的事情了。那么如何区分这两种属性（该祝福的属性、该诅咒的属性）则需要数据来说话。第 3 章将进行更深入的讨论。

本书涵盖的算法与上述论文中提到的其他算法的比较结果如表 1-2 所示。针对表 1-1 列出的问题，表 1-2 列出了性能打分排前 5 名的算法。本书涵盖的算法脱颖而出（提升决策树（Boosted Decision Trees）、随机森林（Random Forests）、投票决策树（Bagged Decision Trees）和逻辑回归（Logistic Regression）。前 3 个属于集成方法。在那篇论文撰写期间惩罚回归还没有获得很好的发展，因此在论文中没有对此进行评价。逻辑回归是属于与之相对比较接近的算法，可以用来评测回归算法的性能。对于论文中的 9 个算法各有 3 种数据规约方法，所以一共是 27 种组合。前 5 名大概占据性能评分排名的前 20%。第 1 行针对 Covt 数据集的算法排名可以看到：提升决策树算法占第 1 名、第 2 名；随机森林算法占第 4 名、第 5 名；投票决策树算法占第 3 名。出现在前 5 名但是本书没有涵盖的算法在最后一列列出（“其他”列）。表中列出的算法包括：K 最近邻（K Nearest Neighbors, KNNs）、人工神经网络（artificial neural nets, ANNs）、支持向量机（support vector machine, SVMs）。

表 1-2　本书涵盖的机器学习算法针对不同问题的性能表现

算法	提升决策树	随机森林	投票决策树	逻辑回归	其他
Covt	1, 2	4, 5	3		
Adult	1, 4	2	3, 5		
LTR.P1	1				支持向量机 K 最近邻
LTR.P2	1, 2	4, 5			支持向量机
MEDIS		1, 3		5	人工神经网络
SLAC		1, 2, 3	4, 5		

续表

算法	提升决策树	随机森林	投票决策树	逻辑回归	其他
HS	1, 3				人工神经网络
MG		2, 4, 5	1, 3		
CALHOUS	1, 2	5	3, 4		
COD	1, 2		3, 4, 5		
BACT	2, 5		1, 3, 4		

在表 1-2 中，逻辑回归只在一个数据集下进入前 5。原因是针对表 1-2 中的数据集规模及其所选的特征，不能体现逻辑回归的优势。相对于数据集的规模（每个数据集有 5 000 个示例）采用的特征太少了（最多也就 200 个）。选取的特征有限，因此现有的数据规模足以选出一个合适的预测模型，而且训练数据集规模又足够小，使得训练时间不至太长，因此其他算法也可以获得很好的结果。

注意 正如将在第 3 章、第 5 章和第 7 章中所看到的那样，当数据含有大量的特征，但是没有足够多的数据或时间来训练更复杂的集成方法模型时，惩罚回归方法将优于其他算法。

Caruana 等人的最新研究（2008 年）关注在特征数量增加的情况下，上述算法的性能。也就是说这些算法面对大数据表现如何。有很多领域的数据拥有的特征已远远超过了第一篇论文中的数据集的规模。例如，基因组问题通常有数以万计的特征（一个基因对应一个特征），文本挖掘问题通常有几百万个特征（每个唯一的词或词对对应一个特征）。线性回归和集成方法随着特征增加的表现如表 1-3 所示。表 1-3 列出了第 2 篇论文中涉及的算法的评分情况，包括算法针对每个问题的性能得分，最右列是此算法针对所有问题的平均得分。算法分成 2 组，上半部分是本书涵盖的算法，下半部分是其他算法。表 1-3 中的算法依次为：BSTDT（Boosted Decision Tress）- 提升决策树；RF（Random Forests）- 随机森林；BAGDT（Bagged Decision Trees）- 投票决策树；BSTST（Boosted Stumps）- 提升二叉树；LR（Logistic Regression）- 逻辑回归；SVM（Support Vector Machines）- 支持向量机；ANN（Artificial Neural Nets）- 人工神经网络；KNN（Distance Weighted kNN）- 距离加权 K 最近邻；PRC（Voted Perceptrons）- 表决感知器；NB（Naive Bayes）- 朴素贝叶斯。NB（Naive Bayes）- 朴素贝叶斯。

表 1-3 中的问题是依其特征规模依次排列的，从 761 个特征到最终的 685569 个特征。线性（逻辑）回归在 11 个测试中的 5 个进入前 3。而且这些优异的分数主要集中在更大规模的数据集部分。注意提升决策树（表 1-3 标为 BSTDT）和随机森林（表 1-3 标为 RF）其表现仍然接近最佳。它们针对所有问题的平均得分排名第 1、第 2。

表 1-3　本书涵盖的机器学习算法面对大数据问题的性能

特征维度	761 STURN	761 CALAM	780 DIGITS	927 TIS	1344 CRYST	3448 KDD98	20958 R-S	105354 CITE	195203 DSE	405333 SPAM	685569 IMDB	MEAN
BSTDT	8	1	2	6	1	3	8	1	7	6	3	1
RF	9	4	3	3	2	1	6	5	3	1	3	2
BAGDT	5	2	6	4	3	1	9	1	6	7	3	4
BSTST	2	3	7	7	7	1	7	4	8	8	5	7
LR	4	8	9	1	4	1	2	2	2	4	4	6
SVM	3	5	5	2	5	2	1	1	5	5	3	3
ANN	6	7	4	5	8	1	4	2	1	3	3	5
KNN	1	6	1	9	6	2	10	1	7	9	6	8
PRC	7	9	8	8	7	1	3	3	4	2	2	9
NB	10	10	10	10	9	1	5	1	9	10	7	10

本书涵盖的算法除了性能外，在其他方面也有优势。惩罚线性回归模型一个重要优势就是它训练所需时间。当面对大规模的数据时，训练所需时间就成为一个需要考量的因素。某些问题的模型训练可能需要几天到几周，这往往是不能忍受的，特别是在开发早期，需要尽早在多次迭代之后找到最佳的方法。惩罚线性回归方法除了训练时间特别快，部署已训练好的模型后进行预测的时间也特别快，可用于高速交易、互联网广告的植入等。研究表明惩罚线性回归在许多情况下可以提供最佳的答案，在即使不是最佳答案的情况下，也可以提供接近最佳的答案。

而且这些算法使用十分简单，可调参数不多，都有定义良好、结构良好的输入数据类型。它们可以解决回归和分类的问题。当面临一个新问题的时候，在 1 ～ 2 小时内完成输入数据的处理、训练模型、输出预测结果是司空见惯的。

这些算法的一个最重要特性就是可以明确地指出哪个输入变量（特征）对预测结果最重要。这已经成为机器学习算法一个无比重要的特性。在预测模型构建过程中，最消耗时间的一步就是特征提取（feature selection）或者叫作特征工程（feature engineering）。就是数据科学家选择哪些变量用于预测结果的过程。根据对预测结果的贡献程度对特征打分，本书涵盖的算法在特征提取过程中可以起到一定的辅助作用，这样可以抛掉一些主观臆测的东西，让预测过程更有一定的确定性。

1.2 什么是惩罚回归方法

惩罚线性回归方法是由普通最小二乘法（ordinary least squares，OLS）衍生出来的。而普通最小二乘法是在大约 200 年前由高斯（Gauss）和法国数学家阿德里安 - 马里 · 勒让德（Legendre）提出的。惩罚线性回归设计之初的想法就是克服最小二乘法的根本缺陷。最小二乘法的一个根本问题就是有时它会过拟合。如图 1-1 所示，考虑用最小二乘法通过一组点来拟合一条直线。这是一个简单的预测问题：给定一个特征 x，预测目标值 y。例如，可以是根据男人的身高来预测其收入。根据身高是可以稍微预测男人的收入的（但是女人不行）。

图 1-1 中的点代表（男人的身高、男人的收入），直线代表使用最小二乘法的预测结果。在某种意义上说，这条直线就代表了在已知男人身高的情况下，对男人收入的最佳预测模型。现在这个数据集有 6 个点。假设这个数据集只有 2 个点。想象有一组点，就像图 1-1 中的点，但是你不能获得全部的点。可能的原因是要获得所有这些点的代价太昂贵了，就像前面提到的基因组数据。只要有足够多的人手，就可以分离出犯罪分子的基因组，但主要问题是代价的原因，你不可能获得他们的全部基因序列。

以简单的例子来模拟这个问题，想象只给你提供当初 6 个点中的任意 2 个点。那么拟

合出来的直线会发生哪些变化？这将依赖于你得到的是哪 2 个点。实际看看这些拟合的效果，可以从图 1-1 中任意选出 2 个点，然后想象穿过这 2 个点的直线。图 1-2 展示了穿过图 1-1 中 2 个点的可能的直线。可以注意到拟合出来的直线依赖于这 2 个点是如何选择的。

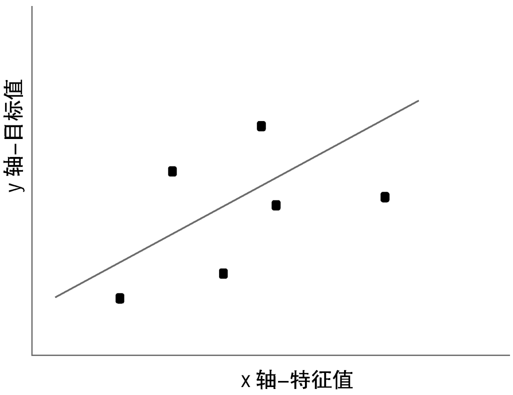


图 1-1　普通最小二乘法拟合

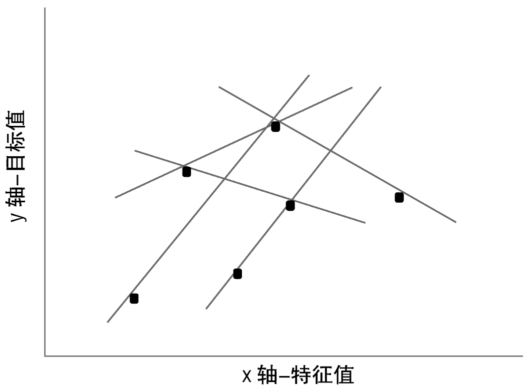


图 1-2　只有 2 个点的情况下拟合的直线

使用 2 个点来拟合一条直线的主要问题是针对直线的自由度（degrees of freedom）[①] 没有提供足够的数据。一条直线有 2 个自由度。有 2 个自由度意味着需要 2 个独立的参数才能唯一确定一条直线。可以想象在一个平面抓住一条直线，然后在这个平面上下滑动这条直线，或者旋转它以改变其斜率。与 x 轴的交点和斜率是相互独立的，它们可以各自改变，两者结合在一起确定了一条直线。一条直线的自由度可以表示成几种等价的方式（可以表示成与 y 轴的交点和斜率、直线上的 2 个点，等等）。所有这些确定一条直线的表示方法都需要 2 个参数。

当自由度与点数相同时，预测效果并不是很好。连接这些点构成了直线，但是在不同点对之间可以形成大量不同的直线。对在自由度与点数相同的情况下所做的预测并不能报太大的信心。图 1-1 是 6 个点拟合一条直线（2 个自由度）。也就是说 6 个点对应 2 个自由度。从大量的人类基因中找到可导致遗传基因的问题可以阐明相似的道理。例如要从大约 20000 个人类基因中找到可导致遗传的基因，可选择的基因越多，需要的数据也越多。20000 个不同基因就代表 20000 个自由度，甚至从 20000 个人获取的数据都不足以得到可靠的结果，在很多情况下，一个相对预算合理的研究项目只能负担得起大约 500 个人的样本数据。在这种情况下，惩罚线性回归就是最佳的选择了。

惩罚线性回归可以减少自由度使之与数据规模、问题的复杂度相匹配。对于具有大量自由度的问题，惩罚线性回归方法获得了广泛的应用。在下列问题中更是得到了偏爱：

[①] 自由度：统计学上的自由度（degree of freedom, df）是指当以样本的统计量来估计总体的参数时，样本中独立或能自由变化的自变量的个数称为该统计量的自由度。

基因问题，通常其自由度（也就是基因的数目）是数以万计的；文本分类问题，其自由度可以超过百万。第 4 章将提供更多的细节：这些方法如何工作、通过示例代码说明算法的机制、用 Python 工具包实现一个机器学习系统的过程示例。

1.3 什么是集成方法

本书涵盖的另一类算法就是集成方法（ensemble methods）。集成方法的基本思想是构建多个不同的预测模型，然后将其输出做某种组合作为最终的输出，如取平均值或采用多数人的意见(投票)。单个预测模型叫作基学习器(base learners)。计算学习理论(computation learning theory）的研究结果证明只要基学习器比随机猜测稍微好些（如果独立预测模型的数目足够多)，那么集成方法就可以达到相当好的效果。

研究人员注意到某些机器学习算法输出结果不稳定，这一问题导致了集成方法的提出。例如，在现有数据集基础上增加新的数据会导致最终的预测模型或性能突变。二元决策树和传统的神经网络就有这种不稳定性。这种不稳定性会导致预测模型性能的高方差，取多个模型的平均值可以看作是一种减少方差的方法。技巧在于如何产生大量的独立预测模型，特别是当它们都采用同样的基学习器时。第 6 章将深入讨论这是如何做到的。方法很巧妙，理解其运作的基本原理也相对容易。下面是其概述。

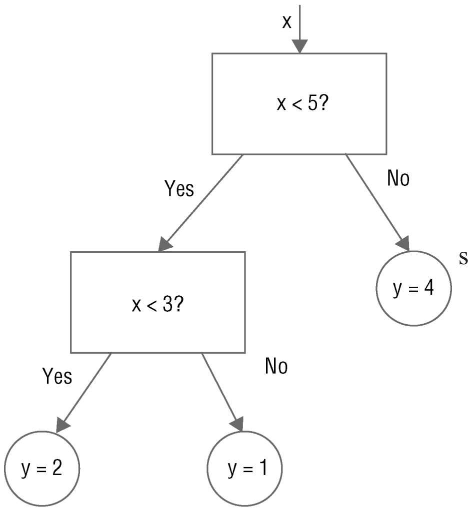


图 1-3 二元决策树示例

集成方法为了实现最广泛的应用通常将二元决策树作为它的基学习器。二元决策树通常如图 1-3 所示。图 1-3 中的二元决策树以一个实数 x 作为最初的输入，然后通过一系列二元（二值）决策来决定针对 x 的最终输出应该是何值。第 1 次决策是判断 x 是否小于 5。如果回答“no”，则二元决策树输出值 4，在由决策的方框下面标为“No”的分支引出的圆圈内。每个可能的 x 值通过决策树都会产生输出 y。图 1-4 将输出 y 画为针对决策树的输入 x 的函数。

由此产生的问题是：这些比较值是如何产生的（如例子中的 $x<5?$），输出的值是如何确定的（图 1-3 决策树底部圆圈中的值）。这些值都来自于基于输入数据的二元决策树的训练。训练算法不难理解，在第 6 章会详细叙述。需要注意的很重要的一点是给定输入数据，训练所得的二元决策树的这些值都是确定的。一种获得不同模型的方法是先对训练数据随

机取样，然后基于这些随机数据子集进行训练。这种技术叫作投票［bagging，来自于自举集成算法（bootstrap aggregating）的简化说法］。此方法可以产生大量的具有稍许差异的二元决策树。这些决策树的输出经过平均或投票产生最终的结果。第 6 章将描述此项技术的细节和其他更有力的工具。

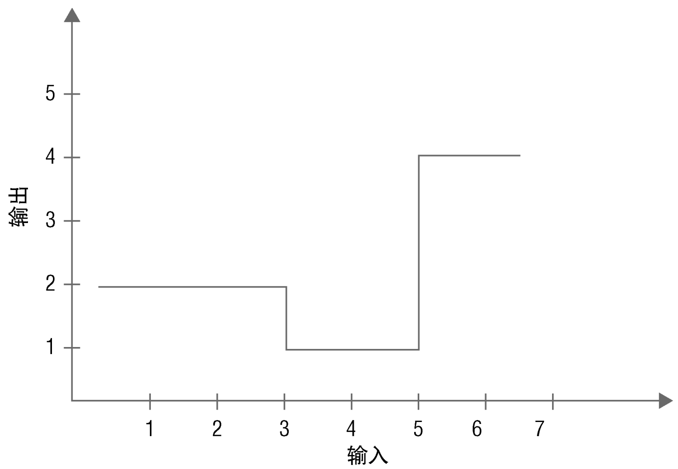


图 1-4 二元决策树示例输入 - 输出图

1.4 算法的选择

这 2 类算法的概要比较如表 1-4 所示。惩罚线性回归的优势在于训练速度非常快。大规模数据集的训练时间可以是小时、天，甚至是几周。要获得一个可以部署的解决方案往往需要进行多次训练。过长的训练时间会影响大数据问题的解决进度及其部署。训练所需时间当然越短越好，因此惩罚线性回归因其训练所需时间短而获得广泛使用就是显而易见的了。依赖于问题，此类算法相比集成方法可能会有一些性能上的劣势。第 3 章将更深入地分析哪类问题适用于惩罚回归，哪类问题适用于集成方法。即使在某些情况下，惩罚线性回归的性能不如集成方法，它也可以是建立一个机器学习系统的有意义的第一步尝试。

表 1-4 惩罚线性回归与集成方法权衡比较

	训练速度	预测速度	问题复杂度	处理大量特征
惩罚线性回归	+	+	−	+
集成方法	−	−	+	−

在系统开发的早期阶段，为了特征的选择、进一步明确问题的形式化描述，训练的过程往往需要多次迭代。决定哪些特征作为预测模型的输入是需要考虑的。有时这个过程是显而易见的，但是通常需要多次迭代之后才逐渐显现出来。把能找到的所有特征都输入进去通常不是一个好的解决方案。

试错法是确定模型最佳输入的典型方法。例如，如果想预测网站的用户是否会点击某个广告链接，首先用到用户的人口统计学信息。但是结果可能并不能达到想要的精度，因此尝试导入用户在此网站过去行为的信息：在过去的网站访问过程中，此用户点击过哪些广告或购买过哪些产品。增加用户访问此网站之前的其他网站的相关信息也会有些帮助。这些尝试都导致了一系列的实验：导入新的数据，然后看看新的数据对结果是否有帮助。这种迭代过程在 2 个方面都是很耗时的：数据的处理、预测模型的训练。惩罚线性回归通常要比集成方法快，而这种时间上的差异性是机器学习系统开发阶段需要考虑的一个重要因素。

例如，如果训练集合在 GB 级别，惩罚线性回归算法的训练时间在 30 分钟这个级别，集成方法可能需要 5 ～ 6 小时。如果特征工程阶段需要 10 次迭代来选择最佳特征集合，则单单这个阶段就会产生 1 天对应 1 周的时间差异。一个有用的技巧就是在开发的早期阶段，如特征工程阶段，利用惩罚线性模型进行训练。这给数据科学家提供一个基本的判断：哪些变量（特征）是有用的、重要的，同时提供了一个后续与其他算法性能比较上的基线。

除了可以获得训练时间上的收益，惩罚线性方法产生预测结果也比集成方法快得多。产生预测结果需要使用一个训练好的模型。对于惩罚线性回归，训练好的模型就是一系列实数：每个实数对应一个用于做预测的特征。所涉及的浮点操作的次数就是用来做预测的变量数。对于对时间高度敏感的预测，如高速交易、互联网广告植入，计算时间上的差异往往意味着盈利还是亏损。

对于一些问题，线性方法相比集成方法可以获得同等或更好的性能。一些问题不需要复杂的模型。第 3 章将详细讨论问题的复杂度，数据科学家的任务就是如何平衡问题的复杂度、预测模型的复杂度和数据集规模，以获得一个最佳的可部署模型。基本思想是如果问题不是很复杂，而且不能获得足够多的数据，则线性方法比更加复杂的集成方法可能会获得全面更优的性能。基因组数据就是此类问题的典型代表。

一般的直观感受是基因数据规模巨大。当然以比特为单位，基因数据集确实是非常庞大的，但是如果为了产生准确的预测，则其规模还需要进一步增加。为了理解两者之间的差别，考虑下面一个假想的实验。假设有 2 个人，一个人有可遗传条件基因，另外一个人没有。如果有这 2 个人的基因序列，那么能确定哪个基因是可遗传条件基因？显然，

这是不可能的，因为这 2 个人之间有很多基因是不同的。那么需要多少人才能完成这个任务呢？至少人数要与基因数相等，如果考虑到噪声，就需要更多的人了。人类大约有 20000 个基因，因计算方法不同而略有差异。获得每条数据大约需要 1000 美元，要获得足够多的数据以完美地解决此问题至少需要 2000 万美元。

就像本章前面讨论的那样，这种情况与用 2 个点来拟合一条直线非常相似。模型的自由度要比数据点少。数据集规模通常需要是自由度的倍数关系。因为数据集的规模是固定的，所以需要调整模型的自由度。惩罚线性回归的相关章节将介绍惩罚线性回归如何支持这种调整以及依此如何达到最优的性能。

注意 本书涵盖的两大类算法的分类与作者和 Jeremy Howard 在 2012 年 O'Reilly Strata 国际会议中提出的完全吻合。Jeremy 负责介绍集成方法，作者负责介绍接受惩罚线性回归，并就两者的优缺点进行了有趣的讨论。事实上，这两类算法占当前构建的预测模型的 80%，这不是没有原因的。

第 3 章将更详细地讨论为什么一个算法或者另一个算法是一个问题的更好选择。这与问题的复杂度、算法内在固有的自由度有关。线性模型倾向于训练速度快，并且经常能够提供与非线性集成方法相当的性能，特别是当能获取的数据受限时。因为它们训练时间短，在早期特征选取阶段训练线性模型是很方便的，然后可以据此大致估计针对特定问题可以达到的性能。线性模型可以提供关于特征对预测的相关信息，可以辅助特征选取阶段的工作。在有充足数据的情况下，集成方法通常能提供更好的性能，也可以提供相对间接的关于结果的贡献的评估。

1.5 构建预测模型的流程

使用机器学习需要几项不同的技能。一项就是编程技能，本书不会把重点放在这。其他的技能用于获得合适的模型进行训练和部署。这些其他技能将是本书重点关注的。那么这些其他技能包括哪些内容？

最初，问题是用多少有些模糊的日常语言来描述的，如“给网站访问者展示他们很可能点击的链接”。将其转换为一个实用的系统需要用具体的数学语言对问题进行重述，找到预测所需的数据集，然后训练预测模型，预测网站访问者对出现的链接点击的可能性。对问题用数学语言进行重叙，其中就包含了对可获得的数据资源中抽取何种特征以及对这些特征如何构建的假设。

当遇到一个新问题时，应该如何着手？首先，需要浏览可获得的数据，确定哪类数据可能用于预测。“浏览”的意思是对数据进行各种统计意义上的检测分析，以获得直观感

受这些数据透露了什么信息，这些信息又与要预测的有怎样的关系。在某种程度上，直觉可以指导你做些工作，也可以量化结果，测试潜在的这些预测特征与结果的相关性。第 2 章将详细介绍对数据集测试分析的过程，本书余下部分所述的算法及其比较会用到这些数据集。

假设通过某种方法，选择了一组特征，开始训练机器学习算法。这将产生一个训练好的模型，然后是估计它的性能。下一步，可能会考虑对特征集进行调整，包括增加新的特征，删除已证明没什么帮助的特征，或者选择另外一种类型的训练目标（也叫作目标函数），通过上述调整看看能否提高性能。可以反复调整设计决策来提高性能。可能会把导致性能比较差的数据单独提出来，然后尝试是否可以从中发现背后的规律。这可以导致添加新的特征到预测模型中，也可以把数据集分成不同的部分分别考虑，分别建立不同的预测模型。

本书的目的是让你熟悉上述处理过程，以后遇到新问题就可以独立完成上述步骤。当重述问题、提取特征、训练算法、评估算法时，需要熟悉不同算法所要求的输入数据结构。此过程通常包括如下步骤。

（1）提取或组合预测所需的特征。

（2）设定训练目标。

（3）训练模型。

（4）评估模型在测试数据上的性能表现。

注意 在完成第一遍过程后，可以通过选择不同的特征集、不同的目标等手段来提高预测的性能。

机器学习要求不仅仅是熟悉一些工具包。它是开发一个可以实际部署的模型的全部过程，包括对机器学习算法的理解和实际的操作。本书的目标就是在这方面提供帮助。本书假设读者具有大学本科的基础数学知识、理解基本的概率和统计知识，但是本书不预设读者具有机器学习的背景知识。同时本书倾向于给读者直接提供针对广泛问题具有最佳性能的算法，而不需要通览所有机器学习相关的算法或方法。有相当数量的算法很有趣，但是因为各种原因并没有获得广泛使用。例如，这些算法可能扩展性不好，不能对内部的运行机理提供直观的解释，或者很难使用，等等。例如，众所周知随机森林算法（本书将会介绍）在在线机器学习算法竞争中遥遥领先。通常有非常切实的原因导致某些算法被经常使用，本书的目标就是在你通读完本书后对这方面具有充分了解。

1.5.1 构造一个机器学习问题

参加机器学习算法竞赛可以看作是解决真实机器学习问题的一个仿真。首先机器学习

算法竞赛会提供一个简短的描述（例如，宣称一个保险公司想基于现有机动车保险政策更好地预测保费损失率）。作为参赛选手，你要做的第一步就是仔细审视数据集中的数据，确定需要做哪种形式的预测。通过对数据的审视，可以获得直观的感受：这些数据代表什么，它们是如何与预测任务关联起来的。数据通常可以揭示可行的方法。图 1-5 描述了从通用语言对预测目标的描述，到对数据的整理准备，以作为机器学习算法输入的基本步骤。

首先，通俗的说法“获得更好的结果”需要先转换成可测量可优化的具体目标。作为网站的拥有者，更好的结果可以是提高点击率或更高的销售额（或更高的利润）。下一步就是收集数据，只要其有助于预测：特定用户有多大可能性会点击各种不同类型的链接，或购买在线提供的各种商品。将这些数据表示为特征的矩阵，如图 1-5 所示。以网站为例，这些特征可能包括：网站访问者之前浏览的其他网页、访问者之前购买的商品。除了用于预测的这些特征，针对此类问题的机器学习算法还需要已知正确的答案用于训练。在图 1-5 中表示为“目标”。本书涵盖的算法通过用户过去的行为来发现用户的购买模式，当然算法不是单纯地记忆用户过去的行为，毕竟一个用户不可能重复购买他昨天刚刚购买的商品。第 3 章将详细讨论无记忆行为的预测模型的训练过程。

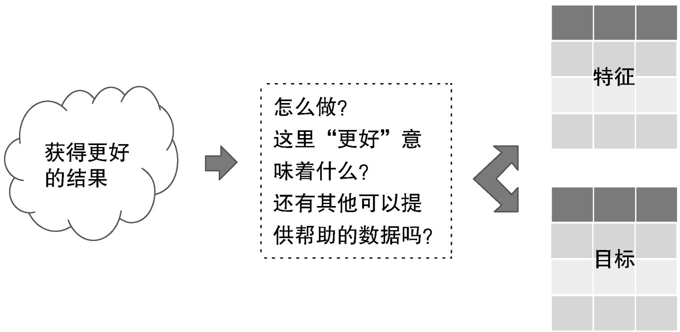


图 1-5　构造一个机器学习问题

通常构造一个机器学习问题可以采用不同的方法。这就导致了问题的构造、模型的选择、模型的训练、模型性能评估这一过程会发生多次迭代，如图 1-6 所示。

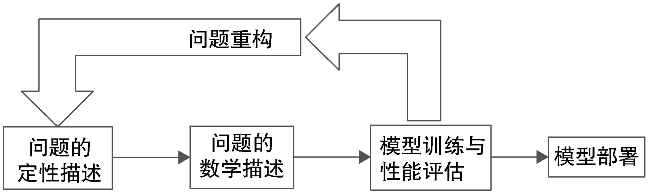


图 1-6　从问题形式化到性能评估的迭代过程

与问题随之而来的是定量的训练目标，或者部分任务是数据提取（这些数据叫作目标或标签）。例如，考虑建立一个自动化预测证券交易的系统。为了实现交易的自动化，第一步可能是预测证券的价格变化。这些价格是很容易获得的，因此利用历史数据构建一个训练模型来预测未来价格的变化应该是容易的。但是即使这一过程包含了多种算法的选择和实验，未来价格的变化仍然可以用多种方法来计算。这种价格的变化可以是当前价格与 10 分钟之后的价格的差异、当前价格与 10 天之后的价格差异，也可以是当前价格与接下来的 10 分钟内价格的最高值、最低值之间的差异。价格的变化可以用一个 2 值的变量来表示："高"或"低"，这依赖于 10 分钟之后价格是升高还是降低。所有这些选择将会导致不同的预测模型，这个预测模型将用于决定是买入还是卖出证券，需要实验来确定最佳的选择。

1.5.2 特征提取和特征工程

确定哪些特征可用于预测也需要实验尝试。这个过程就是特征提取和特征工程。特征提取就是一个把自由形式的各种数据（如一个文档中的字词、一个网页中的字词）转换成行、列形式的数字的过程。例如，垃圾邮件过滤的问题，输入就是邮件的文本，需要提取的东西包括：文本中大写字母的数量、所有大写的词的数量、在文档中出现词"买"的次数，等等，诸如此类的数值型特征。然后基于这些特征把垃圾邮件从非垃圾邮件中区分出来。

特征工程就是对特征进行整理组合，以达到更富有信息量的过程。建立一个证券交易系统包括特征提取和特征工程。特征提取将决定哪些特征可以用来预测价格。过往的价格、相关证券的价格、利率、从最近发布的新闻提取的特征都是现有公开讨论的各种交易系统的输入数据。而且证券的价格还有一系列的工程化特征，包括：指数平滑异同移动平均线（moving average convergence and divergence，MACD）、相对强弱指数（relative strength index，RSI）等。这些特征都是过往价格的函数，它们的发明者都认为这些特征对于证券交易是非常有用的。

选好一系列合理的特征后，就像本书描述的那样，需要训练一个预测模型，评价它的性能，然后决定是否部署此模型。为了确保模型的性能足够满足要求，通常需要调整采用的特征。一个确定使用哪些特征的方法就是尝试所有的组合，但是这样时间代价太大。不可避免地，你面临着提高性能的压力，但是又需要迅速获得一个训练好的模型投入使用。本书讨论的算法有一个很好的特征，它们提供对每个特征对最终预测结果的贡献的度量。经过一轮训练，将会对特征打分以标识其重要性。这些信息可以帮助加速特征工程的过程。

注意 数据准备和特征工程估计会占开发一个机器学习模型 80% ～ 90% 的时间。

模型的训练也是一个过程，每次开始都是先选择作为基线的特征集合。作为一个现

代机器学习算法（如本书描述的算法），通常训练 100 ～ 5000 个不同的模型，然后从中精选出一个模型进行部署。产生如此之多的模型的原因是提供不同复杂度的模型，这样可以挑选出一个与问题、数据集最匹配的模型。如果不想模型太简单又不想放弃性能，不想模型太复杂又不想出现过拟合问题，那么需要从不同复杂度的模型中选择一个最合适的。

1.5.3 确定训练后模型的性能

一个模型合适与否是由此模型在测试数据集上的表现来决定的。这个虽然概念上很简单，却是非常重要的一步。需要留出一部分数据，不用于训练，用于模型的测试。在训练完成之后，用这部分数据集测试算法的性能。本书讨论了留出这部分测试数据的方法。不同的方法各有其优势，主要依赖于训练数据的规模。就像字面上理解那么简单，人们持续地提出各种复杂的方法让测试数据“渗入”训练过程。在处理过程的最后阶段，你将获得一个算法，此算法读取数据，产生准确的预测。在这个过程中，你可能需要检测环境条件的变化，这种变化往往会导致潜在的一些统计特性的变化。

1.6 各章内容及其依赖关系

依赖于读者的背景和是否有时间来了解基本原理，读者可以采用不同的方式来阅读本书。图 1-7 为本书各章之间的依赖关系。

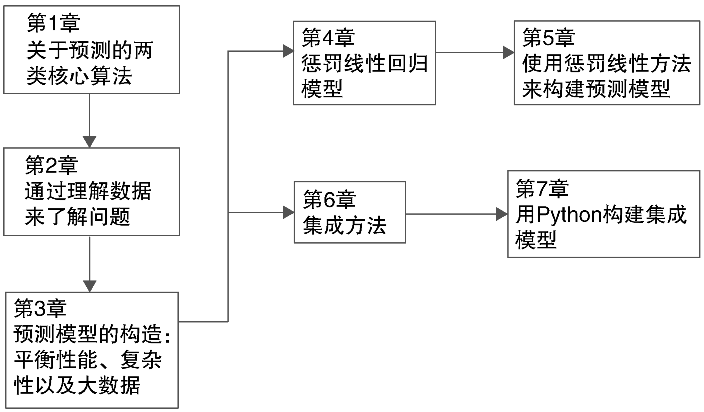


图 1-7 各章依赖关系

第 2 章仔细审视各种数据集。这些数据集用于本书中的问题实例，用以说明算法的使用，以及各种算法之间基于性能和其他特征的比较。面对一个新的机器学习问题的起点就是深入专研数据集，深入理解数据集，了解它们的问题和特质。第 2 章的部分内容就是展示 Python 中可以用于数据集探索的一些工具集。可以浏览第 2 章中的部分例子，不需要

阅读全部例子就可以了解整个流程，当在后续章节遇到具体的应用实例时，可以返回到第 2 章阅读相关的部分。

第 3 章主要介绍机器学习问题中的基本权衡、贯穿本书的关键概念。一个关键概念就是预测问题的数学描述，也会涉及分类和回归问题的差别。第 3 章也介绍了如何使用样本外（out-of-sample）数据（测试数据）来评估预测模型的性能。样本外数据是指在模型训练过程中不包括的数据。一个好的机器学习实践者要求对一个实际部署的预测模型的性能表现有相对稳定的预估。这就要求使用训练数据集以外的数据来模拟新的数据。第 3 章将介绍这么做的原因、实现的方法以及这些方法之间如何取舍。另外一个重要的概念就是系统性能的测量方法，第 3 章将描述这些方法以及它们之间的取舍。对机器学习比较熟悉的读者可以浏览本章，快速略过代码实例，而不需要仔细阅读代码然后运行代码。

第 4 章介绍训练惩罚回归模型的核心思想，以及基本概念及算法的来源。第 3 章引入的一些实例导致了惩罚线性回归方法的产生。第 4 章展示了解决惩罚线性回归训练问题的核心算法代码，以及线性回归方法的几种扩展。一种扩展是将因素变量（factor variable）编码为实数，这样就可以使用线性回归方法。线性回归方法只能用在预测值是数值的情况下，也就是说需要对预测值进行量化。许多实际的重要问题通常的变量是这样的形式：“单身、已婚或离异”等，这种变量对做预测是很有帮助的。如果要引入此种类型的变量（类别变量，categorical variables）到一个线性回归模型，意味着需要设计一种转换方法将类别变量转换为实数变量，第 4 章将会介绍这些方法。第 4 章还介绍叫作基扩展（basis expansion）的方法，此方法从非线性回归中获得非线性函数，有时基扩展用于进一步从线性回归中“挤榨”出一些性能的提升。

第 5 章将第 4 章介绍的惩罚回归算法应用于第 2 章提到的问题中。本章概述实现了惩罚回归算法的 Python 工具包，并用这些工具包来解决问题。本章的目的是尽可能覆盖广泛的各类问题的变体，这样当读者遇到一个问题时，可以找到一个最接近的问题作为借鉴。除了量化并比较预测的性能，第 5 章也考查这些算法的其他特征。理解特征的选择、特征的重要性（对最终预测结果的贡献）是很重要的，这种理解能力可以加快面临新问题时的开发进程。

第 6 章关注集成方法。因为集成方法绝大多数情况下基于二元决策树，第一步就是理解训练和使用二元决策树的原则。集成方法的很多特性都是直接继承于二元决策树。基于上述理解，本章介绍 3 个主要的集成方法：Bagging、提升（boosting）和随机森林。上述每个算法都介绍了使用的基本原则、核心算法的代码，这样读者就会了解如何使用这些算法。

第 7 章应用集成方法来解决第 2 章中的问题，然后对各种算法进行对比分析。对比分

析的内容包括：预测的性能、训练所需的时间和性能等。所有的算法会给出特征的重要性打分。对于特定的问题会对比分析特征在不同的算法中对预测结果的重要性。

以笔者的经验，向程序员和计算机科学家教授机器学习，代码实例要优于数学公式。这就是本书所采用的方法：提供一些基础的数学知识、算法框架和代码实例来说明算法的关键点。本书讨论的几乎所有的算法都可以在本书或网站上找到代码，这么做的初衷就是让读者能够尽快运行代码并解决面临的实际问题。

小结

本章介绍了本书要解决的问题以及构建预测模型的处理流程。本书关注两类算法族。限定介绍的算法的数量，可以让我们更透彻地解释这些算法的背景知识以及这些算法的运行机理。本章通过性能对比说明了为什么选择这两类算法。讨论了这两类算法族的特性和各自的优势，并且详细描述了各自适合解决的问题。

本章还介绍了构建一个预测模型的步骤，每个步骤的各种选择的权衡，对输出结果的考虑。非模型训练时使用的数据可以用来评估预测模型。

本书的目的是使机器学习知之甚少的程序员通过本书的学习，能够胜任将机器学习技术引入项目的工作。本书并不关注大量的算法。相反，只关注当前一流的算法，这些算法可以满足对性能、灵活性和清晰的要求。一旦了解它们是怎么工作的，并且拥有了使用它们的一些经验，就会发现它们很容易上手。这些算法可以解决广泛的问题，而不需要先做大量的训练，这也帮助读者理解这些算法高性能的原因。

参考文献

1. Caruana, Rich, and Alexandru Niculescu - Mizil. “An Empirical Comparison of Supervised Learning Algorithms.” Proceedings of the 23rd International Conference on Machine Learning. ACM, 2006.

2. Caruana, Rich, Nikos Karampatziakis, and Ainur Yessenalina. “An Empirical Evaluation of Supervised Learning in High Dimensions.” Proceedings of the 25th International Conference on Machine Learning. ACM, 2008.

第 2 章

通过理解数据来了解问题

新数据集（问题）就像一个包装好的礼物，它充满了承诺和希望。一旦你能解决它，你就收获了喜悦。但是直到你打开它,它都一直保持着神秘。本章就是告诉你怎么“打开”新的数据集，看清楚里面都有什么，知道如何处置这些数据，并且开始思考如何利用这些数据构建相应的模型。

本章有两个目的：一是熟悉这些数据集，这些数据集被用来作为解决各种类型问题的例子，主要是利用第 4 章和第 6 章介绍的算法；另一个目的就是展示 Python 中分析数据的工具包。

本章用一个简单的例子来回顾基础问题的架构、术语、机器学习数据集的特性。此节介绍的术语将在本书后续章节中用到。在了解了通用的术语后，本章将会依次介绍几类不同的函数逼近问题。这些问题阐明了机器学习问题的通常变体，这样就知道如何识别这些变化，并且知道如何处理它们（本节提供代码实例）。

2.1 “解剖”一个新问题

本书介绍的算法通常是从一个充满了数字，可能是特征（变量）的矩阵（或表格）开始的。表 2-1 展示了一些术语，代表了一个小规模的二维机器学习数据集。此表提供了一个数据集的基本印象，这样对“列代表属性特征，行代表实例”等约定就比较熟悉。这个例子中的问题是预测下一年在线购买书籍所需花费的金额。

表 2-1　　一个机器学习问题的数据

用户 id	属性 1	属性 2	属性 3	标签
001	6.5	Male	12	120 美元
004	4.2	Female	17	270 美元
007	5.7	Male	3	75 美元
008	5.8	Female	8	600 美元

数据是按照行和列组织的。每行代表一个实例（或者叫一个例子、观察）。在表 2-1 中每列指定相应的列名，用来指明在一个机器学习问题中所起的作用。标明为“属性”的列用来预测在买书上所花的费用。在标明为“标签”的列，可以看到去年每个顾客在购书上的花费。

注意 机器学习数据集通常列对应一个属性，行对应一个观察，但也有例外。例如，有些文本挖掘问题的数据矩阵就是另外的形式：列对应一个观察，行对应一个属性。

在表 2-1 中，一行代表一个顾客，此行的数据都与此顾客相关。第一列叫作 UserID（用户 ID），是每行惟一的识别符。实际问题中可能有惟一识别符也可能没有。例如，网站通常为网站的访客建立一个相应的用户 ID，并且在此用户访问网站期间，用户的所有行为都与此用户 ID 绑定。如果用户在此网站上没有注册，则用户的每次访问都将获得一个不同的用户 ID。通常每个观察会被分配一个 ID，这个就是预测的目标对象。第 2 ～第 4 列称为属性，以代替更具体的名字，如身高、性别等。这主要是为了突出他们在预测过程中起到的作用。属性就是在具体实例中用来预测的数据。

标签就是需要预测的数据。在这个例子中，用户 ID 就是一个简单的数字，属性 1 是身高，属性 2 是性别，属性 3 是此人去年阅读的书籍的数量。标签列上的数字代表每人去年在线购书的花费。那么不同类型的数据分别代表什么样的角色呢？一个机器学习算法是如何利用用户 ID、属性和标签列的呢？最简短的回答就是：忽略用户 ID。使用属性来预测标签。

惟一的用户 ID 只是起到记账的目的，在某些情况下可以根据用户 ID 检索到用户的其他数据。通常机器学习算法并不直接使用用户 ID。属性是挑选出来用于预测的。标签是观察得到的结果，机器学习基于此来构建预测模型。

预测通常不用用户 ID 信息，因为它太特殊了。它一般只属于一个实例。一个机器学习的技巧就是构建的模型要有泛化能力（即可以解决新的实例，而不仅仅是把过去的例子都记下来）。为了达到这个目的，算法必须能够关注到不止一行的数据。一个可能的例外是，如果用户 ID 是数字的，并且是按照用户登录的时间依次进行分配的。这样就指示了用户的登录日期，那么如果用户 ID 比较接近，就证明了用户是在比较接近的时间上登录的，依此为条件可以把用户划分为不同的组。

构建预测模型的过程叫作训练。具体的方法依赖于算法，后续章节会详述，但基本上采用迭代的方式。算法假定属性和标签之间存在可预测的关系，观察出错的情况，做出修正，然后重复此过程直到获得一个相对满意的模型。技术细节后续会介绍，这里只是介绍基本思想。

名字的含义：

属性和标签有不同的名字。机器学习的初学者往往被这些名词迷惑，不同的作者可能会采用不同的名字，甚至一篇文章的段落与段落之间都会采用不同的名字。

属性（用来进行预测的变量）也被称为：

- 预测因子
- 特征
- 独立变量
- 输入

标签通常也被称为：

- 结果
- 目标
- 依赖变量
- 响应

2.1.1 属性和标签的不同类型决定模型的选择

表 2-1 中的属性可以分成 2 类：数值变量、类别（或因素、因子）变量。属性 1（身高）是一个数值变量，也是最常见的属性类型。属性 2 是性别，可以是男性或女性。这种类型的属性叫作类别变量或因素变量。类别变量的一个特点就是不同值之间没有顺序关系。男性 < 女性是没有意义的（噢，忘掉几个世纪的争吵吧）。类别变量可以是 2 值的，如男性和女性，也可以是多值的，如美国的州（AL，AK，AR，…WY）。关于属性还有其他差别（如整数与浮点数），但这些差别对机器学习算法的影响并不像数值变量和类别变量那么大。主要原因是很多机器学习算法只能处理数值变量，不能处理类别变量或因素变量（factor variable）。例如，惩罚回归算法只能处理数值变量，SVM、核方法、K 最近邻也是同样。第 4 章将介绍将类别变量转换为数值变量的方法。这些变量的特性将会影响算法的选择以及开发一个预测模型的努力的方向，因此这也是当面临一个新问题时，需要考虑的因素之一。

这种二分法同样适用于标签。表 2-1 所示的标签是数值的：去年在线购书所花费的金额。然而在其他问题中，标签就可能是类别的。例如，如果表 2-1 的任务是预测哪些人下一年的花费超过 200 美元，那么问题就变了，解决问题的方法也随之变了。预测哪些顾客的花费会超过 200 美元的新问题会产生新的标签。这些标签会在两个值中选一个。表 2-1 中的标签与新的逻辑命题“花费 >200 美元”下的新标签之间的关系如表 2-2 所示。表 2-2

所示的新标签取值为：真或假。

表 2-2 数值标签与类别标签

表 2-1 标签	>200 美元？
120 美元	False
270 美元	True
75 美元	False
600 美元	True

当标签是数值的，就叫作回归问题。当标签是类别的，就叫作分类问题。如果分类结果只取 2 个值，就叫作二元分类问题。如果取多个值，就是多类别分类问题。

在很多情况下，问题的类型是由设计者选择的。刚刚的例子就是如何把一个回归问题转换为二元分类问题，只需要对标签做简单的变换。这实际上是面临一个问题时所做的一种权衡。例如，分类目标可以更好地支持 2 种行为选择的决策问题。

分类问题也可能比回归问题简单。例如考虑 2 个地形图的复杂度差异，一个地形图只有一个等高线（如 30.5 米的等高线），而另一个地形图每隔 3.05 米就有一个等高线。只有一个等高线的地形图将地图分成高于 30.5 米的区域和低于 30.5 米的区域，因此相比另一个地形图含有更少的信息。一个分类器就相当于只算出一个等高线，而不再考虑与这条分界线的远近距离之类的问题，而回归的方法就相当于要绘制一个完整的地形图。

2.1.2 新数据集的注意事项

初始审视数据集时，还需要考查数据集的其他特性。下面是一个检查清单，是为了熟悉数据集需要考察的一系列事情，这也有利于明确后续预测模型的开发流程。这些都是很简单的事情，但是直接影响后续步骤，通过这个过程可以了解此数据集的特性。

需要检查的事项：

- 行数、列数
- 类别变量的数目、类别的取值范围
- 缺失的值
- 属性和标签的统计特性

第一个要确认的就是数据的规模。将数据读入二维数组，则外围数组的维度就是行数，内部数组的维度就是列数。下节将会展示针对某一数据集应用此方法获取数据的规模。

下一步就要确定每行有多少缺失的值。这么一行行处理数据的原因是处理缺失数据最简单的方法就是直接抛弃有缺失数据的行（如至少少了一个值的行）。在很多情况下，这样做会导致结果偏差，但在抛弃的行不多的情况下，并不会产生实质性的差异。通过计算

具有缺失数据的行数（加上具体缺失的项数），就可以知道如果采用最简单的方法，则实际上抛弃了多少数据。

如果你有大量的数据，例如正在收集互联网上的数据，那么丢失的数据相对于你获得的数据总量应该是微不足道的。但如果处理的是生物数据，这些数据都比较昂贵，而且有多种属性，这时抛弃这些数据的代价就太大了。在这种情况下，需要找到方法把丢失的值填上，或者使用能够处理丢失数据的算法。把丢失的数据填上的方法一般叫作遗失值插补（imputation）。遗失值插补的最简单方法就是用每行所有此项的值的平均值来代替遗失的值。更复杂的方法要用到第 4 章和第 6 章介绍的预测模型。用预测模型时，将含有遗失值的那列属性当作标签，当然在进行这步之前要确保将初始问题的标签移除。

接下来的小节将从头到尾介绍分析数据集的完整过程，并引入刻画数据集的一些方法，这些都将帮助你确定如何解决建模的问题。

2.2 分类问题：用声纳发现未爆炸的水雷

此小节将介绍在分类问题上首先需要做的工作。首先是简单的测量：数据的规模、数据类型、缺失的数据等。接着是数据的统计特性、属性之间的关系、属性与标签之间的关系。本节的数据集来自 UC Irvine 数据仓库（见参考文献 1）。数据来源于实验：测试声纳是否可以用于检测在港口军事行动后遗留下来的未爆炸的水雷。声纳信号又叫作啁啾信号（chirped signal），即信号在一个脉冲期间频率会增加或降低。此数据集的测量值代表声纳接收器在不同地点接收到的返回信号，其中在大约一半的例子中，返回的声纳信号反映的是岩石的形状，而另一半是金属圆筒的形状（水雷）。下文就用“岩石 vs. 水雷”来代表这个数据集。

2.2.1 “岩石 vs. 水雷”数据集的物理特性

对新数据集所做的第一件事就是确定数据集的规模。代码清单 2-1 为获取“岩石 vs. 水雷”数据集规模的代码。在本章的后续内容，将多次遇到此数据集，主要用来作为介绍算法的例子，此数据集来源于 UC Irvine 数据仓库。在此例中，确定数据集的行数、列数的代码十分简单。数据集文件是由逗号分割的，一次实验数据占据文本的一行。文件处理十分简单：读入一行，对数据按逗号进行分割，将结果列表存入输出列表即可。

代码清单 2-1 确定新数据集规模 -rockVmineSummaries.py
（输出：outputRocksVMinesSummaries.txt）

```
__author__ = 'mike_bowles'
import urllib2
```

```
import sys

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []
labels = []
for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

sys.stdout.write("Number of Rows of Data = " + str(len(xList)) + '\n')
sys.stdout.write("Number of Columns of Data = " + str(len(xList[1])))


Output:
Number of Rows of Data = 208
Number of Columns of Data = 61
```

如代码输出所示，此数据集为 208 行，61 列（每行 61 个字段）。这有什么影响吗？数据集的规模（行数、列数）至少在以下几个方面会影响你对数据的处理。首先，根据数据的规模可以大致判断训练所需的时间。对于像“岩石 vs. 水雷”这种小数据集，训练时间会少于 1 分钟，这有利于在训练过程中不断调整和迭代。如果数据集规模增加到 1 000 × 1 000，惩罚线性回归训练时间将不到一分钟，而集成方法训练时间需要几分钟。如果数据集的行、列增加到万级规模，则惩罚线性回归的训练时间将达到 3 ～ 4 小时，而集成方法则长达 12 ～ 24 小时。更长的训练时间将会影响你的开发进度，因为通常需要迭代几次来对算法进行调整或优化。

另外一个重要的观察是如果数据集的列数远远大于行数，那么采用惩罚线性回归的方法则有很大的可能获得最佳的预测，反之亦然。在第 3 章有实际的例子，这会加深对这个结论的理解。

根据应做事项清单，下一步要做的就是确定哪些列是数值型的，哪些列是类别型的。代码清单 2-2 为针对“岩石 vs. 水雷”数据集完成上述分析的代码。代码依次检查每一列，

确定数值型（整型或浮点型）的条目数量、非空字符串的条目数量、内容为空的条目数量。分析的结果是：前 60 列都是数值型，最后一列都是字符串。这些字符串值是标签。通常类别型变量用字符串表示，如此例所示。在某些情况下，二值类别变量可以表示成 0 和 1。

代码清单 2-2　确定每个属性的特征 -rockVmineContents.py（输出：outputRocksVMinesContents.txt）

```
__author__ = 'mike_bowles'
import urllib2
import sys

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

data = urllib2.urlopen(target_url)


#arrange data into list for labels and list of lists for attributes
xList = []
labels = []
for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)
nrow = len(xList)
ncol = len(xList[1])

type = [0]*3
colCounts = []

for col in range(ncol):
    for row in xList:
        try:
            a = float(row[col])
            if isinstance(a, float):
                type[0] += 1
        except ValueError:
            if len(row[col]) > 0:
```

```
                type[1] += 1
            else:
                type[2] += 1

    colCounts.append(type)
    type = [0]*3

sys.stdout.write("Col#" + '\t' + "Number" + '\t' +
                 "Strings" + '\t ' + "Other\n")
iCol = 0
for types in colCounts:
    sys.stdout.write(str(iCol) + '\t\t' + str(types[0]) + '\t\t' +
                     str(types[1]) + '\t\t' + str(types[2]) + "\n")
    iCol += 1

Output:
Col# Number Strings Other
 0    208     0       0
 1    208     0       0
 2    208     0       0
 3    208     0       0
 4    208     0       0
 5    208     0       0
 6    208     0       0
 7    208     0       0
 8    208     0       0
 9    208     0       0
10    208     0       0
11    208     0       0
 .      .     .       .
 .      .     .       .
 .      .     .       .
54    208     0       0
55    208     0       0
56    208     0       0
57    208     0       0
58    208     0       0
59    208     0       0
60    0     208       0
```

2.2.2 “岩石 vs. 水雷”数据集统计特征

确定哪些属性是类别型，哪些是数值型之后，下一步就是获得数值型属性的描述性统计信息和类别型属性具体类别的数量分布。代码清单 2-3 为这两个处理过程的实例代码。

代码清单 2-3 数值型和类别型属性的统计信息 -rVMSummaryStats.py（输出：outputSummaryStats.txt）

```
__author__ = 'mike_bowles'
import urllib2
import sys
import numpy as np

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)


#arrange data into list for labels and list of lists for attributes
xList = []
labels = []

for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)
nrow = len(xList)
ncol = len(xList[1])

type = [0]*3
colCounts = []

#generate summary statistics for column 3 (e.g.)
col = 3
colData = []
for row in xList:
    colData.append(float(row[col]))

colArray = np.array(colData)
```

```
colMean = np.mean(colArray)
colsd = np.std(colArray)
sys.stdout.write("Mean = " + '\t' + str(colMean) + '\t\t' +
            "Standard Deviation = " + '\t ' + str(colsd) + "\n")

#calculate quantile boundaries
ntiles = 4

percentBdry = []

for i in range(ntiles+1):
    percentBdry.append(np.percentile(colArray, i*(100)/ntiles))

sys.stdout.write("\nBoundaries for 4 Equal Percentiles \n")
print(percentBdry)
sys.stdout.write(" \n")

#run again with 10 equal intervals
ntiles = 10

percentBdry = []

for i in range(ntiles+1):
    percentBdry.append(np.percentile(colArray, i*(100)/ntiles))

sys.stdout.write("Boundaries for 10 Equal Percentiles \n")
print(percentBdry)
sys.stdout.write(" \n")


#The last column contains categorical variables

col = 60
colData = []
for row in xList:
    colData.append(row[col])

unique = set(colData)
sys.stdout.write("Unique Label Values \n")
print(unique)
```

```
#count up the number of elements having each value

catDict = dict(zip(list(unique),range(len(unique))))

catCount = [0]*2

for elt in colData:
    catCount[catDict[elt]] += 1
sys.stdout.write("\nCounts for Each Value of Categorical Label \n")
print(list(unique))
print(catCount)

Output:
Mean =   0.053892307       Standard Deviation =       0.046415983

Boundaries for 4 Equal Percentiles
 [0.0057999999999999996, 0.024375000000000001, 0.044049999999999999,
 0.064500000000000002, 0.4264]

Boundaries for 10 Equal Percentiles
[0.00579999999999, 0.0141, 0.022740000000, 0.0278699999999,
0.0362200000000, 0.0440499999999, 0.050719999999, 0.0599599999999,
0.0779400000000, 0.10836, 0.4264]
Unique Label Values
set(['R', 'M'])

Counts for Each Value of Categorical Label
['R', 'M']
[97, 111]
```

代码第一部分读取数值型数据的某一列，然后产生它的统计信息。第一步计算此属性的均值和方差。了解这些统计信息可以加强在建立预测模型时的直观感受。

第二部分代码主要是为了找到异常值。基本过程如下:假设在下面数值列表 [0.1,0.15，0.2，0.25，0.3，0.35，0.4，4] 中确定是否有异常值，显然最后一个数“4”是异常值。

发现这种异常值的一种方法是：将一组数字按照百分位数进行划分。例如，第 25 百分位数是含有最小的 25% 的数，第 50 百分位数是含有最小的 50% 的数。把这种分组可视化最简单的方法是假想把这些数据按顺序排列。上述的例子已经按顺序排好，这样就可

以很容易地看到百分位数的边界。一些经常用到的百分位数通常被赋予特殊的名字。将数组按照 1/4、1/5、1/10 划分的百分位数通常分别叫作四分位数（quartiles，按顺序排列的一组数据被划分为 4 个相等部分的分割点的数值）、五分位数（quintiles）和十分位数（deciles）。

上述的数组很容易定义出四分位数，因为此数组已按顺序排好，共有 8 个元素。第一个四分位数含有 0.1 和 0.15，以下的以此类推。可以注意到这些四分位数的跨度。第一个是 0.05（0.15 ～ 0.1）。第二个四分位数的跨度也大致相同。然而最后一个四分位数的跨度却是 3.6，这个是其他四分位数跨度的几十倍。

代码清单 2-3 中四分位数边界的计算过程与之类似。程序计算四分位数，然后显示最后一个四分位数的跨度要比其他的宽很多。为了更加准确，又计算了十分位数，同样证明了最后一个十分位数的跨度要远远大于其他的十分位数。有些情况下的最后一个分位数变宽是正常的，因为通常数据的分布在尾部会变稀疏。

2.2.3 用分位数图展示异常点

更具体地研究异常点（异常值）的一个方法就是画出数据的分布图，然后与可能的分布进行比较，判断相关的数据是否匹配。代码清单 2-4 展示如何使用 Python 的 probplot 函数来帮助确认数据中是否含有异常点。分布图展示了数据的百分位边界与高斯分布的同样百分位的边界对比。如果此数据服从高斯分布，则画出来的点应该是一条直线。来自“岩石 vs. 水雷”数据集的第 4 列（第 4 属性）的一些点远离这条直线，如图 2-1 所示。这说明此数据集尾部的数据要多于高斯分布尾部的数据。

代码清单 2-4 “岩石 vs. 水雷”数据集的第 4 列的分位数图 -qqplotAttribute.py

```
__author__ = 'mike bowles'
import numpy as np
import pylab
import scipy.stats as stats
import urllib2
import sys

target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

data = urllib2.urlopen(target_url)
```

```
#arrange data into list for labels and list of lists for attributes
xList = []
labels = []

for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)
nrow = len(xList)
ncol = len(xList[1])
type = [0]*3
colCounts = []

#generate summary statistics for column 3 (e.g.)
col = 3
colData = []
for row in xList:
    colData.append(float(row[col]))


stats.probplot(colData, dist="norm", plot=pylab)
pylab.show()
```

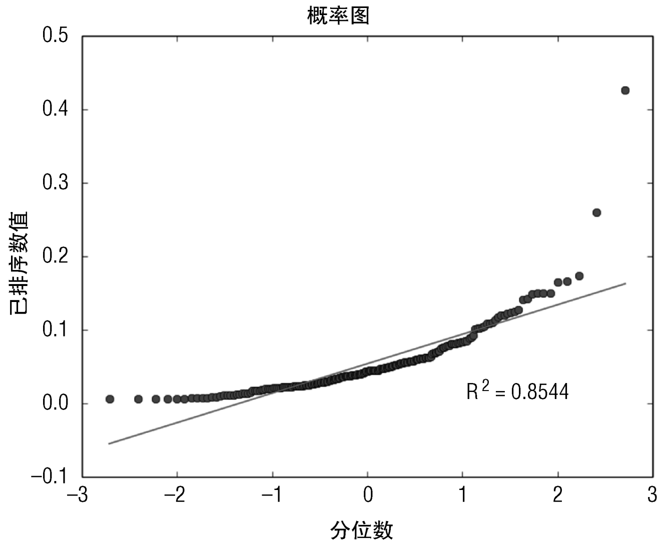


图 2-1　“岩石 vs. 水雷”数据集第 4 属性的分位数图

那么如何利用这些信息？异常点在建模或预测期间都会带来麻烦。基于此数据集训练完一个模型后，可以查看此模型预测错误的情况，然后确认此错误是否与这些异常点有关。如果确实是这样的话，可以采取步骤进行校正。例如，可以复制这些预测模型表现不好的例子，以加强这些例子在数据集中的比重。也可以把这些不好的例子分离出来，然后单独训练。如果认为预测模型在真正部署时不会遇到此类异常数据，则也可以把这些例子排除出数据集。一个可行办法是在对数据集进行探究阶段，先产生四分位数边界，然后看看潜在的异常点的规模对后续建模及预测可能的影响。这样在分析错误时，可以通过分位数图（quantile-quantile，Q-Q）确定哪些数据可以称为异常点。

2.2.4 类别属性的统计特征

上述的分析过程只适用于数值属性。那么类别属性呢？你可能想知道一共可以分为几类、每类数据的数目。想获得这些信息主要是基于以下原因：性别属性有两个值（男、女），但是如果属性是美国的州，则有 50 个可能的值。随着属性数目的增加，处理的复杂度也在增加。绝大多数二元决策树算法（集成方法的基础）对于其可以处理的类别数是有限制的。由 Breiman 和 Cutler（此算法的发明人）写的流行的随机森林算法包支持 32 个类别。如果一个属性超过 32 个类别，则需要合并。

有时在训练过程中会随机抽取数据集的一个子集，然后在此子集上训练一系列的模型。例如，如果类别属性就是美国的州，其中爱达荷州只出现了两次。一个随机抽取的训练用数据子集中很可能不含有爱达荷州的样本。你需要在这些问题发生前就预见到可能会出现这样的情况，然后再着手进行处理。以两个爱达荷州的样本为例，可以把它与蒙大纳州或怀俄明州合并，也复制这两个样本（增加其所占的比例），或者控制随机取样保证抽取到含有爱达荷州的样本，这个过程叫作分层抽样（stratified sampling）。

2.2.5 利用 Python Pandas 对“岩石 vs. 水雷”数据集进行统计分析

Python Pandas 工具包可以帮助自动化数据统计分析的过程，已经被证实在数据预处理阶段特别有用。Pandas 工具包可以将数据读入一种特定的数据结构，叫作数据框（data frame）。数据框是依据 CRAN-R 数据结构建模的。

注意 Pandas 工具包的安装可能会有困难，主要原因是它有一系列的依赖，每个依赖必须安装正确的版本，而且相互之间要匹配，或者诸如此类的问题。绕过此类障碍的一个简单的方法就是直接安装 Anaconda Python Distribution 分发包，此分发包可以直接从 Continuum Analytics（http://continuum.io）处下载。安装过程十分简单，只要按指令依次进行就可以安装好数据分析、机器学习所需的大量软件包。

你可以把数据框当成一个表格或者类似矩阵的数据结构，如表 2-1 所示。数据框定义行代表一个实例（一次实验、一个例子、一次测量等），列代表一个特定的属性。此结构像矩阵，但又不是矩阵，因为每列的元素很可能是不同类型的。形式上矩阵里的所有元素都是来自一个域的（如实数、二进制数、复数等）。但对于统计学来说，矩阵的限制太严格了，因为统计方面的一个样本往往是多个不同类型的值的混合。

表 2-1 样例中的第 1 个属性列是实数，第两个属性列是类别变量（属性），第 3 个属性列是整数。在一个列内，所有元素的取值都是同一类型，但是列与列之间是不同的。通过数据框，可以通过索引（index）的方式访问具体某个元素，类似 Python 中访问一个 Numpy 数组或二维数组中的元素（element）。类似地，采用索引切片（index slicing）可以访问整行或整列，而且在 Pandas 数据框中，可以通过名字来访问行或列。这对于小规模或中等规律的数据是十分方便的（搜索“Pandas introduction”会找到关于使用 Pandas 的入门指导的链接）。

如何从 UC Irvine 数据仓库网站读取“岩石 vs. 水雷”数据的 CSV 文件如代码清单 2-5 所示。这里的输出只是完整输出中的一部分。自行运行代码就可以获得完整输出。

代码清单 2-5 用 Python Pandas 读入数据、分析数据 - pandasReadSummarizer.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#print head and tail of data frame
print(rocksVMines.head())
print(rocksVMines.tail())

#print summary of data frame
summary = rocksVMines.describe()
print(summary)

Output (truncated):

       V0     V1      V2    ...     V57     V58     V59 V60
```

```
0 0.0200 0.0371 0.0428   ... 0.0084 0.0090 0.0032  R
1 0.0453 0.0523 0.0843   ... 0.0049 0.0052 0.0044  R
2 0.0262 0.0582 0.1099   ... 0.0164 0.0095 0.0078  R
3 0.0100 0.0171 0.0623   ... 0.0044 0.0040 0.0117  R
4 0.0762 0.0666 0.0481   ... 0.0048 0.0107 0.0094  R

[5 rows x 61 columns]

        V0     V1     V2   ...    V57    V58    V59 V60
203 0.0187 0.0346 0.0168   ... 0.0115 0.0193 0.0157  M
204 0.0323 0.0101 0.0298   ... 0.0032 0.0062 0.0067  M
205 0.0522 0.0437 0.0180   ... 0.0138 0.0077 0.0031  M
206 0.0303 0.0353 0.0490   ... 0.0079 0.0036 0.0048  M
207 0.0260 0.0363 0.0136   ... 0.0036 0.0061 0.0115  M

[5 rows x 61 columns]


              V0          V1    ...         V58         V59
count 208.000000 208.000000    ... 208.000000 208.000000
mean    0.029164   0.038437    ...   0.007941   0.006507
std     0.022991   0.032960    ...   0.006181   0.005031
min     0.001500   0.000600    ...   0.000100   0.000600
25%     0.013350   0.016450    ...   0.003675   0.003100
50%     0.022800   0.030800    ...   0.006400   0.005300
75%     0.035550   0.047950    ...   0.010325   0.008525
max     0.137100   0.233900    ...   0.036400   0.043900
```

读入数据后，程序第一部分首先打印头数据和尾数据。注意到所有的头数据都有R标签，所有的尾数据都有M标签。对于这个数据集，第一部分是R标签的（岩石），第二部分是M标签的（水雷）。在分析数据时首先要注意到此类信息。在后续章节中会看到，确定模型的优劣有时需要对数据进行取样。那么取样就需要考虑到数据的存储结构。最后的代码打印输出实数属性列的统计信息。

Pandas 可以自动计算出均值、方差、分位数。由于 describe 函数输出的总结（统计信息）本身就是一个数据框，因此可以自动化属性值的筛选过程以发现异常点。可以比较不同分位数之间的差异。对于同一属性列，如果存在某一个差异严重异于其他差异，则说明存在异常点。这就值得进一步探究这些异常点牵扯到多少行数据，这些异常点涉及的数据很可能是少量的，这些都需要仔细分析。

2.3 对“岩石 vs. 水雷”数据集属性的可视化展示

可视化可以提供对数据的直观感受，这个有时是很难通过表格的形式把握到的。此节将介绍很有用的可视化方法。分类问题和回归问题的可视化会有所不同。在有鲍鱼和红酒数据集的章节中看到回归问题的可视化方法。

2.3.1 利用平行坐标图进行可视化展示

对于具有多个属性问题的一种可视化方法叫作平行坐标图（parallel coordinates plot）。图 2-2 为平行坐标图的基本样式。图右边的向量（[1 3 2 4]）代表数据集中某一行属性的值。这个向量的平行坐标图如图 2-2 中的折线所示。这条折线是根据属性的索引值和属性值画出来的。整个数据集的平行坐标图对于数据集中的每一行属性都有对应的一条折线。基于标签对折线标示不同的颜色，更有利于观测到属性值与标签之间的关系。（在 Wikipedia 输入“parallel coordinates”会检索出更多的例子。）

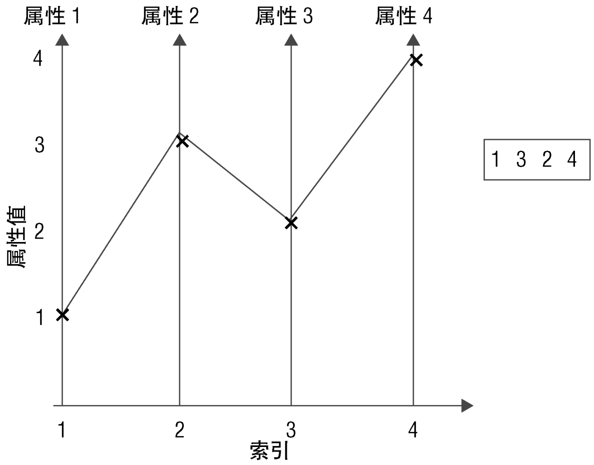


图 2-2 平行坐标图

代码清单 2-6 展示了如何获得“岩石 vs. 水雷”数据集的平行坐标图。图 2-3 为结果。折线根据对应的标签赋予不同的颜色：R（岩石）是蓝色，M（水雷）是红色。有时候图画出来后标签（类别）之间可以很明显地区分出来。著名的“鸢尾花数据集”类别之间就可以很明显地区分出来，这就是机器学习算法进行分类应该达到的效果。对应“岩石 vs. 水雷”数据集则看不到明显的区分。但是有些区域蓝色和红色的折线是分开的。沿着图的底部，蓝色的线要突出一点儿，在属性索引 30 ～ 40 之间，蓝色的线多少要比红色的线高一些。[①] 这些观察将有助于解释和确认某些预测的结果。

代码清单 2-6 实数值属性的可视化：平行坐标图 – linePlots.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
```

① 建议读者自行运行代码，观察输出的彩色平行坐标图，就可以看到文中所述的效果。下面的图例也有同样的问题。——译者注。

```
import matplotlib.pyplot as plot
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

for i in range(208):
    #assign color based on "M" or "R" labels
    if rocksVMines.iat[i,60] == "M":
        pcolor = "red"
    else:
        pcolor = "blue"

    #plot rows of data as if they were series data
    dataRow = rocksVMines.iloc[i,0:60]
    dataRow.plot(color=pcolor)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()
```

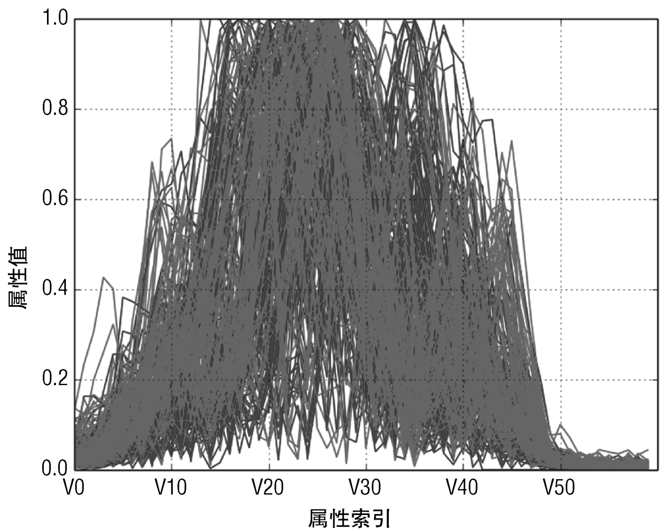


图 2-3 “岩石 vs. 水雷”数据集属性的平行坐标图

2.3.2　属性和标签的关系可视化

另外一个需要了解的问题就是属性之间的关系。获得这种成对关系的快速方法就是绘制属性与标签的交会图（cross-plots)。代码清单 2-7 展示了如何产生代表性属性对的交会图。这些交会图（又叫作散点图，scatter plots）展示了这些属性对之间关系的密切程度。

代码清单 2-7　属性对的交会图 -corrPlot.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#calculate correlations between real-valued attributes
dataRow2 = rocksVMines.iloc[1,0:60]
dataRow3 = rocksVMines.iloc[2,0:60]

plot.scatter(dataRow2, dataRow3)


plot.xlabel("2nd Attribute")
plot.ylabel(("3rd Attribute"))
plot.show()

dataRow21 = rocksVMines.iloc[20,0:60]

plot.scatter(dataRow2, dataRow21)


plot.xlabel("2nd Attribute")
plot.ylabel(("21st Attribute"))
plot.show()
```

图 2-4 和图 2-5 为来自“岩石 vs. 水雷”数据集的两对属性的散点图。“岩石 vs. 水雷”

数据集的属性是声纳返回的取样值。声纳返回的信号又叫啁啾信号 ，因为它是一个脉冲信号，开始在低频，然后上升到高频。这个数据集的属性就是声波由岩石或水雷反射回来的时间上的取样。这些返回的声学信号携带的时间与频率的关系与发出的信号是一样的。数据集的 60 个属性是返回的信号在 60 个不同时间点的取样（因此是 60 个不同的频率）。你可能会估计相邻的属性会比隔一个的属性更相关，因为在相邻时间上的取样在频率上的差别应该不大。

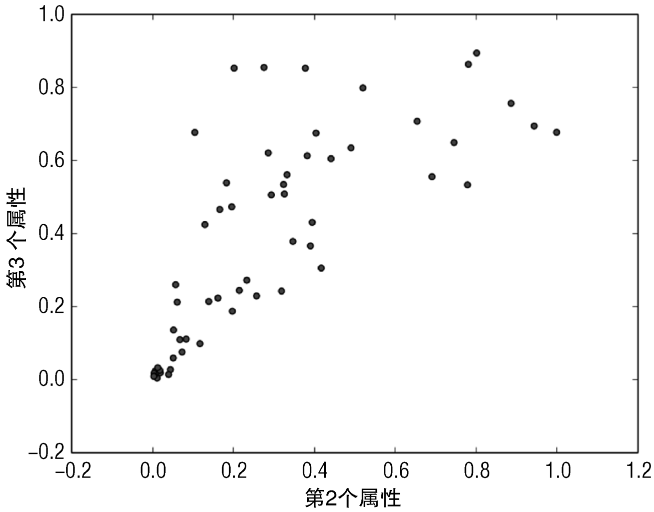


图 2-4 “岩石 vs. 水雷”数据集第 2 个属性与第 3 个属性的交会图

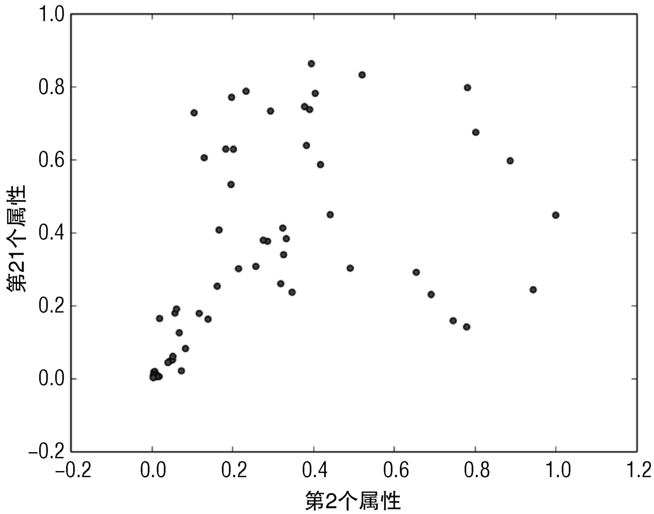


图 2-5 “岩石 vs. 水雷”数据集第 2 个属性与第 21 个属性的交会图

这种直观感受在图 2-4 和图 2-5 中得到了证实。图 2-4 中的点要比图 2-5 中的点更集中于一条直线。如果想进一步了解数值相关和散点图的形状两者之间的关系，请在 wikipedia 搜索“correlation（相关）”相关页面。基本上，如果散点图上的点沿着一条“瘦”直线排列，则说明这两个变量强相关；如果这些点形成一个球型，则说明这些点不相关。

应用同样原则，可以画出任何一个属性与最终目标（标签）的散点图，研究两者之间的相关性。若对应目标是实数（回归问题），则画出的散点图会与图 2-4 和图 2-5 十分相似。“岩石 vs. 水雷”数据集是一个分类问题，目标是二值的，但是遵循同样的步骤。

代码清单 2-8 展示如何画出标签和第 35 个属性的散点图。为什么选用第 35 个属性作为展示属性与标签关系的例子，灵感来自于平行坐标图 2-3。这个平行坐标图显示岩石数据与水雷数据在属性索引值 35 左右有所分离。则标签与索引值 35 附近的属性的关系也应该显示这种分离，正如图 2-6 和图 2-7 所示。

代码清单 2-8　分类问题标签和实数值属性之间的相关性 -targetCorr.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
from random import uniform
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#change the targets to numeric values
target = []
for i in range(208):
    #assign 0 or 1 target value based on "M" or "R" labels
    if rocksVMines.iat[i,60] == "M":
        target.append(1.0)
    else:
        target.append(0.0)

#plot 35th attribute
dataRow = rocksVMines.iloc[0:208,35]
plot.scatter(dataRow, target)
```

```
plot.xlabel("Attribute Value")
plot.ylabel("Target Value")
plot.show()

#
#To improve the visualization, this version dithers the points a little
# and makes them somewhat transparent
target = []
for i in range(208):

#assign 0 or 1 target value based on "M" or "R" labels
    # and add some dither

    if rocksVMines.iat[i,60] == "M":
        target.append(1.0 + uniform(-0.1, 0.1))
    else:
        target.append(0.0 + uniform(-0.1, 0.1))

    #plot 35th attribute with semi-opaque points
dataRow = rocksVMines.iloc[0:208,35]
plot.scatter(dataRow, target, alpha=0.5, s=120)

plot.xlabel("Attribute Value")
plot.ylabel("Target Value")
plot.show()
```

如果把M用 1 代表，R 用 0 代表，就会得到如图 2-6 所示的散点图。在图 2-6 中可以看到一个交会图常见的问题。当其中一个变量只取有限的几个值时，很多点会重叠在一起。如果这种点很多，则只能看到很粗的一条线，分辨不出这些点是如何沿线分布的。

代码清单 2-8 产生了第二图，通过 2 个小技巧克服了上述的问题。每个点都加上一个小的随机数，产生了少量的离散值（这里是对标签值进行了处理）。标签值最初是 0 或 1。在代码中可以看到，标签值加上了一个在 −0.1 和 0.1 之间均匀分布的随机数，这样就把这些点分散开，但是又不至于把这 2 条线混淆。此外，这些点绘制时取 alpha=0.5，这样这些点就是半透明的。那么在散点图中若多个点落在一个位置就会形成一个更黑的区域，这时需要对数据做一些微调使你能看到你想看到的。

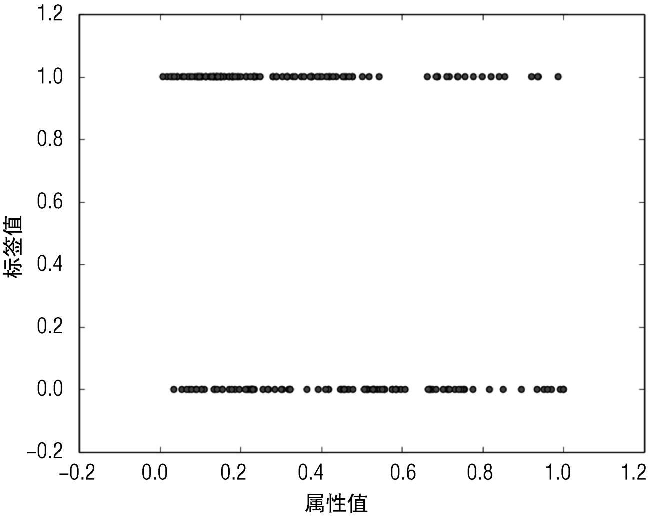


图 2-6 标签 - 属性交会图

这两种方法的效果如图 2-7 所示。可以注意到第 35 个属性在左上方的点更加集中一些，然而在下面的数据从右到左分布得更加均匀些。上方的数据对应水雷的数据。下面的数据对应岩石的数据。由图观察可知，可以因此建立一个分类器，判断第 35 个属性是否大于或小于 0.5。如果大于 0.5，就判断为岩石，如果小于 0.5，就判断为水雷。在第 35 个属性值小于 0.5 的实例中，水雷的分布要更密集，而且在属性值小于 0.5 的实例中，岩石的分布要稀疏得多。这样就可以获得一个比随机猜测好些的结果。

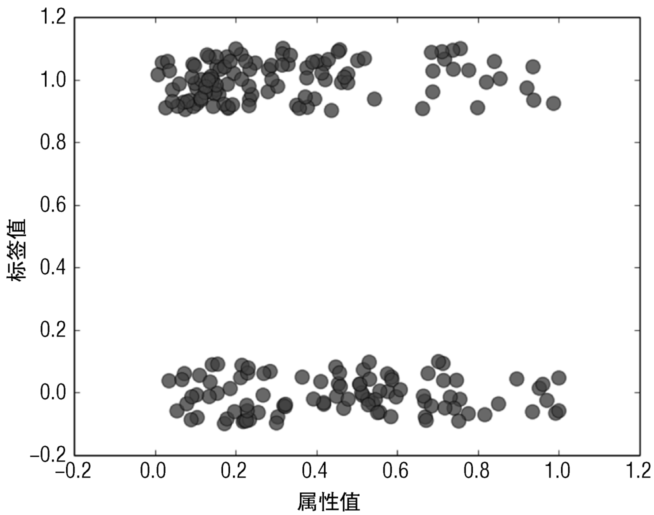


图 2-7 经过扰动和半透明处理的标签 - 属性图

注意 在第 5 章和第 7 章将会看到更系统的构建分类器的方法。它们会用到所有的属性，而不仅仅是一、二个属性。当看到它们是如何做决策时，可以回头看看本章的例子就会理解为什么它们的选择是明智的。

两个属性（或一个属性、一个标签）的相关程度可以由皮尔逊相关系数（Pearson’s correlation coefficient）来量化。给定 2 个等长的向量 u 和 v（如公式 2-1 和公式 2-2 所示）。首先 u 的所有元素都减去 u 的均值（见公式 2-3）。对 v 也做同样的事情。

$$u = \begin{matrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{matrix}$$

公式 2-1　向量 u 的元素

$$\bar{u} = avg(u)$$

公式 2-2　向量 u 的均值

$$\Delta u = \begin{matrix} u_1 - \bar{u} \\ u_2 - \bar{u} \\ \vdots \\ u_n - \bar{u} \end{matrix}$$

公式 2-3　向量 u 中每个元素都减去均值

以向量 Δu 相同的定义方式，对应第二个向量 v，定义向量 Δv。

则 u 和 v 之间的皮尔森相关系数如公式 2-4 所示。

$$corr(u,v) = \frac{\Delta u^T * \Delta v}{\sqrt{(\Delta u^T * \Delta u) * (\Delta v^T * \Delta v)}}$$

公式 2-4　皮尔森相关系数定义

代码清单 2-9 展示了用该函数计算图 2-3 和图 2-5 中属性对的相关系数。相关系数和图中展示的结果一致，索引值距离比较近的属性间相关系数也比较高。

代码清单 2-9　对属性 2 和属性 3、属性 2 和属性 21 分别计算各自的皮尔森相关系数 -corrCalc.py

```
__author__ = 'mike_bowles'
import pandas as pd
```

```
from pandas import DataFrame
from math import sqrt
import sys
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#calculate correlations between real-valued attributes
dataRow2 = rocksVMines.iloc[1,0:60]
dataRow3 = rocksVMines.iloc[2,0:60]
dataRow21 = rocksVMines.iloc[20,0:60]

mean2 = 0.0; mean3 = 0.0; mean21 = 0.0
numElt = len(dataRow2)
for i in range(numElt):
    mean2 += dataRow2[i]/numElt
    mean3 += dataRow3[i]/numElt
    mean21 += dataRow21[i]/numElt

var2 = 0.0; var3 = 0.0; var21 = 0.0
for i in range(numElt):
    var2 += (dataRow2[i] - mean2) * (dataRow2[i] - mean2)/numElt
    var3 += (dataRow3[i] - mean3) * (dataRow3[i] - mean3)/numElt
    var21 += (dataRow21[i] - mean21) * (dataRow21[i] - mean21)/numElt

corr23 = 0.0; corr221 = 0.0
for i in range(numElt):

corr23 += (dataRow2[i] - mean2) * \
              (dataRow3[i] - mean3) / (sqrt(var2*var3) * numElt)
    corr221 += (dataRow2[i] - mean2) * \
               (dataRow21[i] - mean21) / (sqrt(var2*var21) * numElt)

sys.stdout.write("Correlation between attribute 2 and 3 \n")
print(corr23)
sys.stdout.write(" \n")
sys.stdout.write("Correlation between attribute 2 and 21 \n")
print(corr221)
```

```
sys.stdout.write(" \n")


Output:
Correlation between attribute 2 and 3
0.770938121191

Correlation between attribute 2 and 21
0.466548080789
```

2.3.3 用热图（heat map）展示属性和标签的相关性

对于计算少量的相关性，将相关性结果打印输出或者画成散点图都是可以的。但是对于大量的数据，就很难用这种方法整体把握相关性。如果问题有 100 以上的属性，则很难把散点图压缩到一页。

获得大量属性之间相关性的一种方法就是计算出每对属性的皮尔森相关系数后，将相关系数构成一个矩阵，矩阵的第 ij-th 个元素对应第 *i* 个属性与第 *j* 个属性的相关系数，然后把这些矩阵元素画到热图上。代码清单 2-10 为热图的代码实现。图 2-8 就是这种热图。沿着斜对角线的浅色区域证明索引值相近的属性相关性较高。正如上文提到的，这与数据产生的方式有关。索引相近说明是在很短的时间间隔内取样的，因此声纳信号的频率也接近，频率相近说明目标（标签）也类似。

代码清单 2-10 属性相关系数可视化 -sampleCorrHeatMap.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")

#read rocks versus mines data into pandas data frame
rocksVMines = pd.read_csv(target_url,header=None, prefix="V")

#calculate correlations between real-valued attributes
corMat = DataFrame(rocksVMines.corr())

#visualize correlations using heatmap
```

```
plot.pcolor(corMat)
plot.show()
```

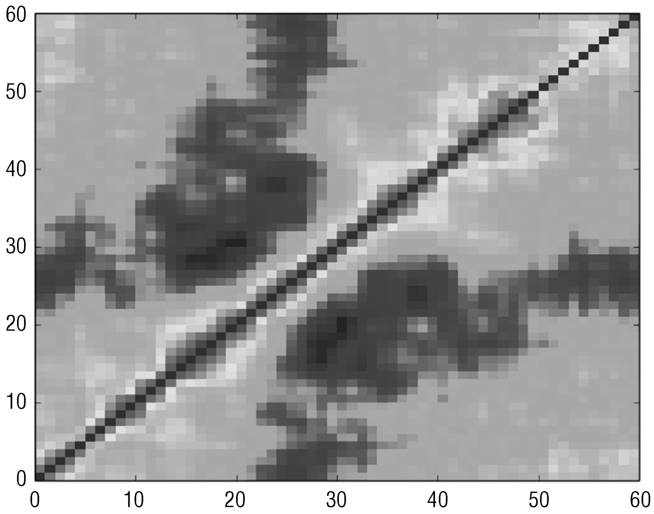


图 2-8 展示属性对相关性的热图

属性之间如果完全相关（相关系数 =1）意味着数据可能有错误，如同样的数据录入两次。多个属性间的相关性很高（相关系数 >0.7），即多重共线性（multicollinearity），往往会导致预测结果不稳定。属性与标签的相关性则不同，如果属性和标签相关，则通常意味着两者之间具有可预测的关系。

2.3.4 对“岩石 vs. 水雷”数据集探究过程小结

在探究“岩石 vs. 水雷”数据集的过程中，本节介绍了一系列的工具，以加深对数据集的理解和直观感受。本节已深入细节，如这些工具的来源、用法等。下节将用同样的工具来分析后续机器学习算法用到的其他数据集。因为这些工具都已介绍过，下一节将只介绍因为问题的不同，而对工具所做的改变。

2.4 基于因素变量的实数值预测：鲍鱼的年龄

探测未爆炸的水雷数据集的工具同样可以用于回归问题。在给定物理测量值的情况下，预测鲍鱼的年龄就是此类问题的一个实例。鲍鱼的属性中包括因素属性，下面将说明属性中含有因素属性后与上例有什么不同。

鲍鱼数据集的问题是根据某些测量值预测鲍鱼年龄。当然可以对鲍鱼进行切片，然后

数年轮获得鲍鱼年龄的精确值，就像通过数树的年轮得到树的年龄一样。但是问题是这种方法代价比较大，耗时（需要在显微镜下数年轮）。因此更方便经济的方法是做些简单的测量，如鲍鱼的长度、宽度、重量等指标，然后通过一个预测模型对其年龄做相对准确的预测。预测分析有大量的科学应用，学习机器学习的一个好处就是可以将其应用到一系列很有趣的问题上。

鲍鱼数据集可以从 UC Irvine 数据仓库中获得，其 URL 是 http://archive.ics.uci.edu/ml/machine-learning-database/abalone/abalone.data。此数据集数据以逗号分隔，没有列头。每个列的名字存在另外一个文件中。代码清单 2-11 将鲍鱼数据集读入 Pandas 数据框，然后进行分析，这些分析与“分类问题：用声纳探测未爆炸的水雷”节中的一样。由数据的性质决定的，“岩石 vs. 水雷”数据集的列名（属性名）更加通用。为了能够从直觉上判断提出的预测模型是否可接受，理解鲍鱼数据集各个列名（属性名）的意义是十分重要的。因此，在代码中将列名（属性名）直接拷贝到代码中，与相关的数据绑定在一起，帮助直接感受下一步机器学习算法应该怎么预测。建立预测模型所需的数据包括性别、长度、直径、高度、整体重量、去壳后重量、脏器重量、壳的重量、环数。最后一列“环数”是十分耗时采获得的，需要锯开壳，然后在显微镜下观察得到。这是一个有监督机器学习方法通常需要的准备工作。基于一个已知答案的数据集构建预测模型，然后用这个预测模型预测不知道答案的数据。

代码清单 2-11 不仅展示了产生统计信息的代码，而且展示了打印输出的统计信息。第一部分打印数据集的头和尾。为了节省空间只显示了头。当你自己运行代码时，就可以看到全部的输出。绝大多数数据框中的数据是浮点数。第一列是性别，标记为 M（雄性）、F（雌性）和 I（不确定的）。鲍鱼的性别在出生时是不确定的，成熟一些之后才能确定。因此对于小的鲍鱼其性别是不确定的。鲍鱼的性别是一个三值的类别变量。类别属性需要特别注意。一些算法只能处理实数值的属性（如支持向量机（support vector machines）、K 最近邻、惩罚线性回归，这些将在第 4 章介绍）。第 4 章会讨论把类别属性转换成实数值属性的技巧。代码清单 2-11 还展示了实数值属性按列的统计信息。

代码清单 2-11　鲍鱼数据集的读取与分析 -abaloneSummary.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot

target_url = ("http://archive.ics.uci.edu/ml/machine-"
              "learning-databases/abalone/abalone.data")
```

```
#read abalone data
abalone = pd.read_csv(target_url,header=None, prefix="V")
abalone.columns = ['Sex', 'Length', 'Diameter', 'Height',
                   'Whole weight','Shucked weight', 'Viscera weight',
                   'Shell weight', 'Rings']


print(abalone.head())
print(abalone.tail())

#print summary of data frame
summary = abalone.describe()
print(summary)

#box plot the real-valued attributes
#convert to array for plot routine
array = abalone.iloc[:,1:9].values
boxplot(array)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges"))
show()

#the last column (rings) is out of scale with the rest
# - remove and replot
array2 = abalone.iloc[:,1:8].values
boxplot(array2)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges"))
show()

#removing is okay but renormalizing the variables generalizes better.
#renormalize columns to zero mean and unit standard deviation
#this is a common normalization and desirable for other operations
# (like k-means clustering or k-nearest neighbors
abaloneNormalized = abalone.iloc[:,1:9]

for i in range(8):
    mean = summary.iloc[1, i]
    sd = summary.iloc[2, i]
```

```
abaloneNormalized.iloc[:,i:(i + 1)] = (
                    abaloneNormalized.iloc[:,i:(i + 1)] - mean) / sd


array3 = abaloneNormalized.values
boxplot(array3)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges - Normalized "))
show()

Printed Output: (partial)
  Sex Length Diameter Height Whole wt Shucked wt Viscera wt
0   M  0.455    0.365  0.095   0.5140     0.2245     0.1010
1   M  0.350    0.265  0.090   0.2255     0.0995     0.0485
2   F  0.530    0.420  0.135   0.6770     0.2565     0.1415
3   M  0.440    0.365  0.125   0.5160     0.2155     0.1140
4   I  0.330    0.255  0.080   0.2050     0.0895     0.0395

   Shell weight Rings
0         0.150    15
1         0.070     7
2         0.210     9
3         0.155    10
4         0.055     7
     Sex Length Diameter Height Whole weight Shucked weight
4172   F  0.565    0.450  0.165       0.8870         0.3700
4173   M  0.590    0.440  0.135       0.9660         0.4390
4174   M  0.600    0.475  0.205       1.1760         0.5255
4175   F  0.625    0.485  0.150       1.0945         0.5310
4176   M  0.710    0.555  0.195       1.9485         0.9455

     Viscera weight Shell weight Rings
4172         0.2390       0.2490    11
4173         0.2145       0.2605    10
4174         0.2875       0.3080     9
4175         0.2610       0.2960    10
4176         0.3765       0.4950    12
           Length    Diameter      Height    Whole wt  Shucked wt
count 4177.000000 4177.000000 4177.000000 4177.000000 4177.000000
mean     0.523992    0.407881    0.139516    0.828742    0.359367
```

```
std        0.120093     0.099240     0.041827     0.490389     0.221963
min        0.075000     0.055000     0.000000     0.002000     0.001000
25%        0.450000     0.350000     0.115000     0.441500     0.186000
50%        0.545000     0.425000     0.140000     0.799500     0.336000
75%        0.615000     0.480000     0.165000     1.153000     0.502000
max        0.815000     0.650000     1.130000     2.825500     1.488000

       Viscera weight Shell weight        Rings
count     4177.000000  4177.000000  4177.000000
mean         0.180594     0.238831     9.933684
std          0.109614     0.139203     3.224169
min          0.000500     0.001500     1.000000
25%          0.093500     0.130000     8.000000
50%          0.171000     0.234000     9.000000
75%          0.253000     0.329000    11.000000
max          0.760000     1.005000    29.000000
```

不仅可以列出统计信息，还可以像代码清单 2-11 那样产生每个实数值属性（列）的箱线图（box plots）。第一个箱线图如图 2-9 所示。箱线图又叫作盒须图（box and whisker plots）、盒式图、盒状图。这些图显示了一个小长方形，有一个红线穿过它。红线代表此列数据的中位数（第 50 百分位数），长方形的顶和底分别表示第 25 百分位数和第 75 百

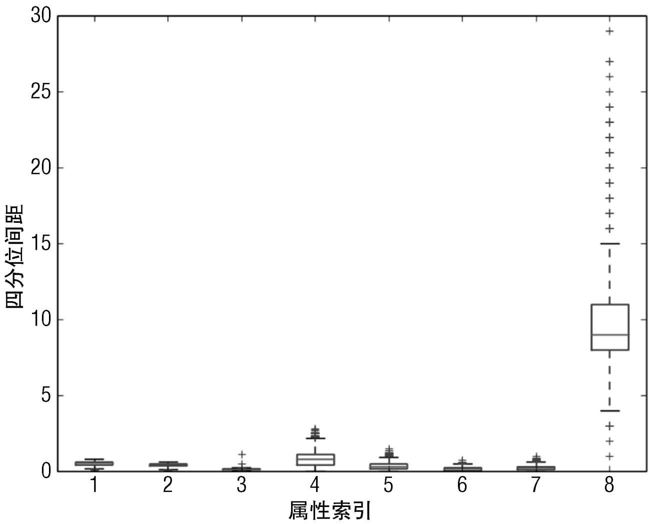


图 2-9 鲍鱼数据集的实数值属性箱线图

分位数（或者第一四分位数、第三四分位数）。可以比较打印出来的统计信息和箱线图中的线段来证实这一点。在盒子的上方和下方有小的水平线，叫作盒须（whisker）。它们分别据盒子的上边和下边是四分位间距的 1.4 倍，四分位间距就是第 75 百分位数和第 25 百分位数之间的距离，也就是从盒子的顶边到盒子底边的距离。也就是说盒子上面的盒须到盒子顶边的距离是盒子高度的 1.4 倍。这个盒须的 1.4 倍距离是可以调整的，详见箱线图的相关文档。在有些情况下，盒须要比 1.4 倍距离近，这说明数据的值并没有扩散到原定计算出来的盒须的位置。在这种情况下，盒须被放在最极端的点上。在另外一些情况下，数据扩散到远远超出计算出的盒须的位置（1.4 倍盒子高度的距离），这些点被认为是异常点。

图 2-9 所示的箱线图是一种比打印出数据更快、更直接的发现异常点的方法，但是最后一个环数属性（最右边的盒子）的取值范围导致其他属性都被“压缩”了（导致很难看清楚）。一种简单的解决方法就是把取值范围最大的那个属性删除。结果如图 2-10 所示。这个方法并不令人满意，因为没有实现根据取值范围自动缩放（自适应）。

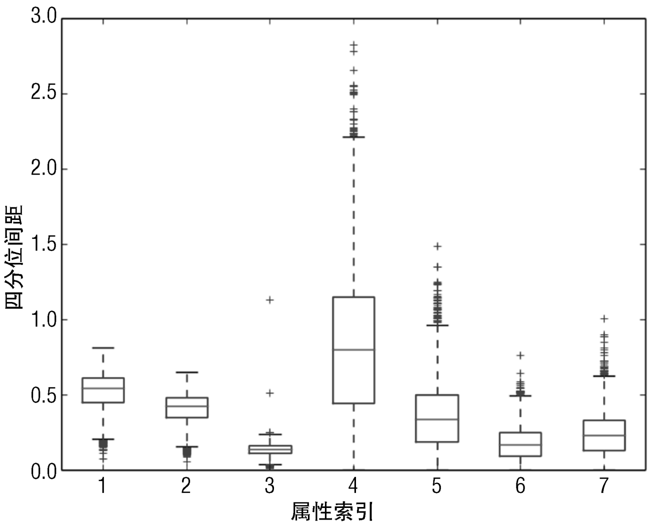


图 2-10 鲍鱼数据集实数值属性箱线图

代码清单 2-11 的最后一部分代码在画箱线图之前将属性值归一化（normalization）。此处的归一化指确定每列数据的中心，然后对数值进行缩放，使属性 1 的一个单位值与属性 2 的一个单位值相同。在数据科学中有相当数量的算法需要这种归一化。例如，K-means 聚类方法是根据行数据之间的向量距离来进行聚类的。距离是对应坐标上的点相减然后取平方和。单位不同，算出来的距离也会不同。到一个杂货店的距离以英里为单位是 1 英里，以英尺为单位就是 5 280 英尺。代码清单 2-11 中的归一化是把属性数值都转换为均值为 0、

标准差为 1 的分布。这是最通用的归一化。归一化计算用到了函数 summary() 的结果。归一化后的效果如图 2-11 所示。

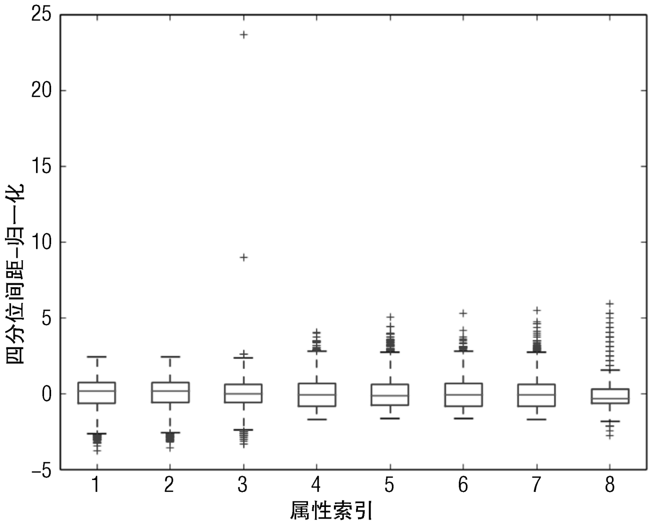


图 2-11 归一化鲍鱼数据集属性的箱线图

注意归一化到标准差 1.0 并不意味着所有的数据都在 −1.0 和 +1.0 之间。盒子的顶边和底边多少都会在 −1.0 和 +1.0 附近，但是还有很多数据在这个边界外。

2.4.1 回归问题的平行坐标图：鲍鱼问题的变量关系可视化

下一步是看属性之间、属性与标签之间的关系。对于“岩石 vs. 水雷”数据集，加颜色的平行坐标图以图形化方式展示了这两种关系。针对鲍鱼问题，上述方法需要做些修正。岩石 vs. 水雷是分类问题。平行坐标图对于此类问题，折线代表了一行数据，折线的颜色表明了其所属的类别。这有利于可视化属性和所属类别之间的关系。鲍鱼问题是一个回归问题，应该用不同的颜色来对应标签值的高低。也就是实现由标签的实数值到颜色值的映射，需要将标签的实数值压缩到 [0.0,1.0] 区间。代码清单 2-12 由函数 summary() 获得最大、最小值实现这种转换。结果如图 2-12 所示。

代码清单 2-12 鲍鱼数据的平行坐标图 -abalonParallelPlot.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot
from math import exp
```

```
target_url = ("http://archive.ics.uci.edu/ml/machine-"
              "learning-databases/abalone/abalone.data")
#read abalone data
abalone = pd.read_csv(target_url,header=None, prefix="V")
abalone.columns = ['Sex', 'Length', 'Diameter', 'Height',
                   'Whole Wt', 'Shucked Wt',
                   'Viscera Wt', 'Shell Wt', 'Rings']
#get summary to use for scaling
summary = abalone.describe()
minRings = summary.iloc[3,7]
maxRings = summary.iloc[7,7]
nrows = len(abalone.index)

for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = abalone.iloc[i,1:8]
    labelColor = (abalone.iloc[i,8] - minRings) / (maxRings - minRings)
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()

#renormalize using mean and standard variation, then compress
# with logit function
meanRings = summary.iloc[1,7]
sdRings = summary.iloc[2,7]

for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = abalone.iloc[i,1:8]
    normTarget = (abalone.iloc[i,8] - meanRings)/sdRings
    labelColor = 1.0/(1.0 + exp(-normTarget))
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()
```

图 2-12 的平行坐标图为鲍鱼年龄（壳的环数）和用于预测年龄的属性之间的关系。折线使用的颜色标尺从深红棕色、黄色、浅蓝色一直到深蓝色。图 2-11 的箱线图显示整个数据集的最大值和最小值分布十分广泛。图 2-12 有压缩的效果，导致绝大多数的数据都分布在颜色标尺的中间部分。尽管如此，图 2-12 还是能够显示每个属性和目标环数的相关性。在属性值相近的地方，折线的颜色也比较接近，则会集中在一起。这些相关性都暗示可以构建相当准确的预测模型。相对于那些体现了良好相关性的属性和目标环数，有些微弱的蓝色折线与深橘色的区域混合在一起，说明有些实例可能很难正确预测。

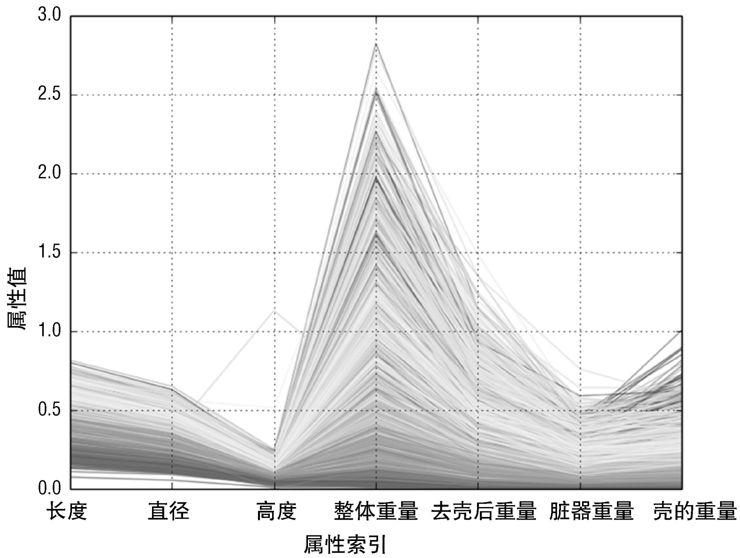


图 2-12 鲍鱼数据集的彩色平行坐标图

改变颜色映射关系可以从不同的层面来可视化属性与目标之间的关系。代码清单 2-11 的最后一部分用到了箱线图中用过的归一化。此归一化不是让所有的值都落到 0 和 1 之间。首先让取负值的数据与取正值的数据基本上一样多。代码清单 2-11 中使用分对数变换（logit transform）实现数值到（0,1）的映射。分对数变换如公式 2-5 所示，分对数函数如图 2-13 所示。

$$\mathrm{logit}\ transform(x) = 1/(1+e^{-x})$$

公式 2-5 分对数转换公式

如图 2-13 所示，分对数函数将很大的负数映射成 0 （接近），很大的正数映射成 1（接近）， 0 映射成 0.5。在第 4 章还会看到分对数函数，在将线性函数与概率联系起来它起到了关键的作用。

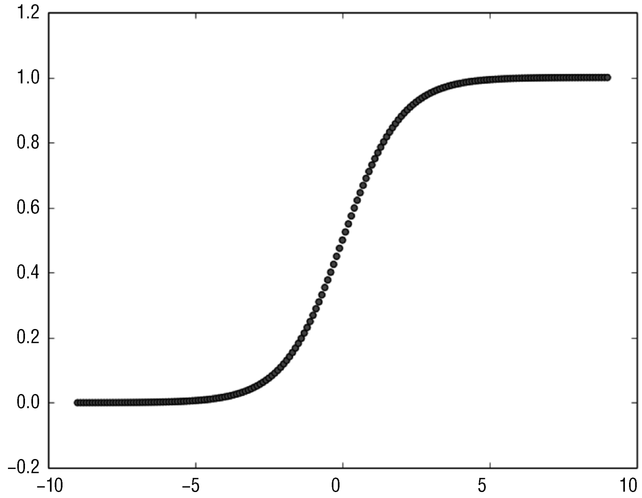


图 2-13 分对数函数

图 2-14 为归一化之后的结果。转换后可以更充分地利用颜色标尺中的各种颜色。注意到针对整体重量和去壳后的重量这两个属性，有些深蓝的线（对应具有大环数的品种）混入了浅蓝线的区域，甚至是黄色、亮红的区域。这意味着，当鲍鱼的年龄较大时，仅仅这些属性不足以准确地预测出鲍鱼的年龄（环数）。好在其他属性（如直径、壳的重量）可以很好地把深蓝线区分出来。这些观察都有助于分析预测错误的原因。

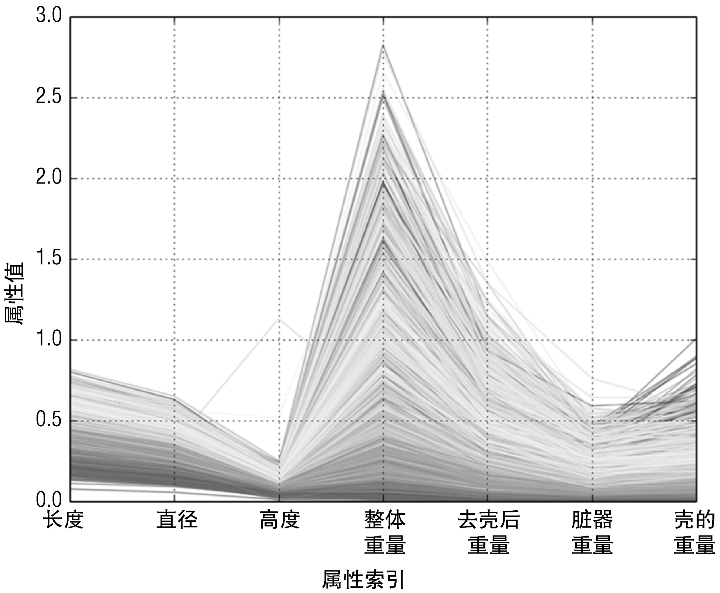


图 2-14 鲍鱼数据的平行坐标图

2.4.2　回归问题如何使用关联热图——鲍鱼问题的属性对关系的可视化

最后一步是看不同属性之间的相关性和属性与目标之间的相关性。代码清单 2-13 为针对鲍鱼数据产生关联热图和关系矩阵的代码。遵循的方法与“岩石 vs. 水雷”数据集相应章节里的方法一样，只有一个重要差异：因为鲍鱼问题是进行实数值预测，所以在计算关系矩阵时可以包括目标值。

代码清单 2-13　鲍鱼数据的相关性计算 -abaloneCorrHeat.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
import matplotlib.pyplot as plot

target_url = ("http://archive.ics.uci.edu/ml/machine-"
              "learning-databases/abalone/abalone.data")
#read abalone data
abalone = pd.read_csv(target_url,header=None, prefix="V")
abalone.columns = ['Sex', 'Length', 'Diameter', 'Height',
                   'Whole weight', 'Shucked weight',
                   'Viscera weight', 'Shell weight', 'Rings']

#calculate correlation matrix
corMat = DataFrame(abalone.iloc[:,1:9].corr())
#print correlation matrix
print(corMat)

#visualize correlations using heatmap
plot.pcolor(corMat)
plot.show()

                 Length Diameter   Height Whole Wt  Shucked Wt
Length         1.000000 0.986812 0.827554  0.925261   0.897914
Diameter       0.986812 1.000000 0.833684  0.925452   0.893162
Height         0.827554 0.833684 1.000000  0.819221   0.774972
Whole weight   0.925261 0.925452 0.819221  1.000000   0.969405
Shucked weight 0.897914 0.893162 0.774972  0.969405   1.000000
Viscera weight 0.903018 0.899724 0.798319  0.966375   0.931961
Shell weight   0.897706 0.905330 0.817338  0.955355   0.882617
Rings          0.556720 0.574660 0.557467  0.540390   0.420884
```

```
                  Viscera weight Shell weight    Rings
Length                  0.903018     0.897706 0.556720
Diameter                0.899724     0.905330 0.574660
Height                  0.798319     0.817338 0.557467
Whole weight            0.966375     0.955355 0.540390
Shucked weight          0.931961     0.882617 0.420884
Viscera weight          1.000000     0.907656 0.503819
Shell weight            0.907656     1.000000 0.627574
Rings                   0.503819     0.627574 1.000000
```

图 2-15 为关联热图。在图 2-15 中，红色代表强相关，蓝色代表弱相关。目标（壳上环数）是最后一项，即关联热图的第一行和最右列。蓝色说明这些属性与目标弱相关。浅蓝对应目标与壳的重量的相关性。这个结果与在平行坐标图看到的一致。如图 2-15 所示，在偏离对角线的单元内，红棕色的值代表这些属性相互高度相关。这多少与平行坐标图的结论有些矛盾，因为在平行坐标图中，目标与属性的一致性是相当强的。代码清单 2-13 展示了具体的关联值。

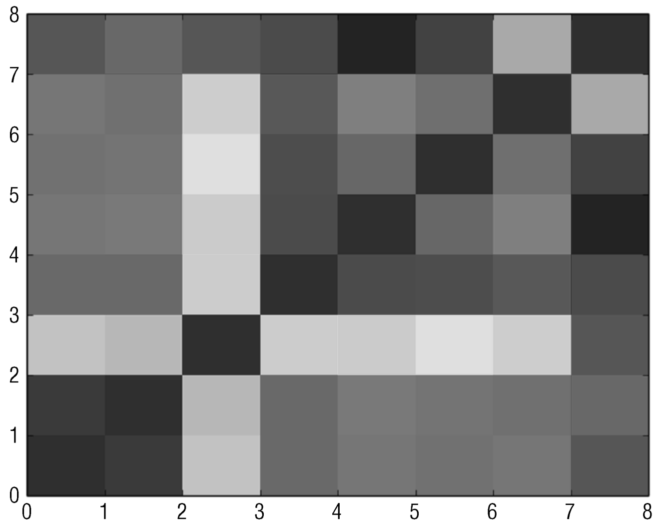


图 2-15 鲍鱼数据的关联热图

在此小节看到如何将用于分类问题（岩石 vs. 水雷）的工具修改后用于回归问题。修改主要来源于这两类问题的本质差别：回归问题标签是实数值，而二元分类问题的标签是二值变量。下节将采用同样的手段来分析全部是数值属性的回归问题。因为是回归问题，

可以采用鲍鱼数据分析同样的工具。因为所有属性都是数值型，分析时可以包含所有的属性，如计算相关性时。

2.5　用实数值属性预测实数值目标：评估红酒口感

红酒口感数据集包括将近 1 500 种红酒的数据。每一种红酒都有一系列化学成分的测量指标，包括酒精含量、挥发性酸、亚硝酸盐。每种红酒都有一个口感评分值，是三个专业评酒员的评分的平均值。问题是构建一个预测模型，输入化学成分的测量值，预测口感评分值，使之与评酒员的评分一致。

代码清单 2-14 为获得红酒数据集统计信息的代码。代码打印输出数据集的数值型统计信息，在代码清单的最后部分可以看到。代码还产生了归一化属性的箱线图，可以直观发现数据集中的异常点。图 2-16 为箱线图。数值型统计信息和箱线图都显示含有大量的边缘点。在对此数据集进行训练时要记住这一点。当分析预测模型的性能时，这些边缘点很可能就是分析模型预测错误的一个重要来源。

代码清单 2-14　红酒数据统计信息 -wineSummary.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot

target_url = ("http://archive.ics.uci.edu/ml/machine-"
"learning-databases/wine-quality/winequality-red.csv")
wine = pd.read_csv(target_url,header=0, sep=";")

print(wine.head())

#generate statistical summaries
summary = wine.describe()
print(summary)

wineNormalized = wine
ncols = len(wineNormalized.columns)

for i in range(ncols):
    mean = summary.iloc[1, i]
```

```
    sd = summary.iloc[2, i]

wineNormalized.iloc[:,i:(i + 1)] = \
        (wineNormalized.iloc[:,i:(i + 1)] - mean) / sd
array = wineNormalized.values
boxplot(array)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges - Normalized "))
show()

Output - [filename - wineSummary.txt]
    fixed acidity volatil acid citric acid resid sugar chlorides
0            7.4          0.70        0.00          1.9     0.076
1            7.8          0.88        0.00          2.6     0.098
2            7.8          0.76        0.04          2.3     0.092
3           11.2          0.28        0.56          1.9     0.075
4            7.4          0.70        0.00          1.9     0.076

    free sulfur dioxide tot sulfur dioxide density   pH sulphates
0                    11                  34  0.9978 3.51      0.56
1                    25                  67  0.9968 3.20      0.68
2                    15                  54  0.9970 3.26      0.65
3                    17                  60  0.9980 3.16      0.58
4                    11                  34  0.9978 3.51      0.56

    alcohol quality
0       9.4       5
1       9.8       5
2       9.8       5
3       9.8       6
4       9.4       5
        fixed acidity volatile acidity citric acid residual sugar
count    1599.000000      1599.000000 1599.000000    1599.000000
mean        8.319637         0.527821    0.270976       2.538806
std         1.741096         0.179060    0.194801       1.409928
min         4.600000         0.120000    0.000000       0.900000
25%         7.100000         0.390000    0.090000       1.900000
50%         7.900000         0.520000    0.260000       2.200000
75%         9.200000         0.640000    0.420000       2.600000
max        15.900000         1.580000    1.000000      15.500000
```

```
       chlorides free sulfur dioxide tot sulfur dioxide     density
count 1599.000000        1599.000000        1599.000000 1599.000000
mean     0.087467          15.874922          46.467792    0.996747
std      0.047065          10.460157          32.895324    0.001887
min      0.012000           1.000000           6.000000    0.990070
25%      0.070000           7.000000          22.000000    0.995600
50%      0.079000          14.000000          38.000000    0.996750
75%      0.090000          21.000000          62.000000    0.997835
max      0.611000          72.000000         289.000000    1.003690

               pH   sulphates     alcohol     quality
count 1599.000000 1599.000000 1599.000000 1599.000000
mean     3.311113    0.658149   10.422983    5.636023
std      0.154386    0.169507    1.065668    0.807569
min      2.740000    0.330000    8.400000    3.000000
25%      3.210000    0.550000    9.500000    5.000000
50%      3.310000    0.620000   10.200000    6.000000
75%      3.400000    0.730000   11.100000    6.000000
max      4.010000    2.000000   14.900000    8.000000
```

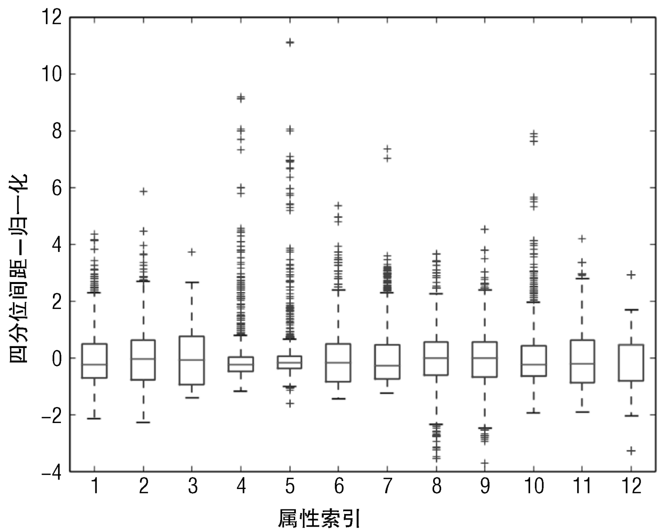


图 2-16 归一化红酒数据的属性与目标箱线图

加入颜色标记的平行坐标图更易于观察属性与目标的相关程度。代码清单 2-15 为生

成平行坐标图的代码。图 2-17 为平行坐标图。图 2-17 的主要不足在于对取值范围较小的变量进行了压缩。

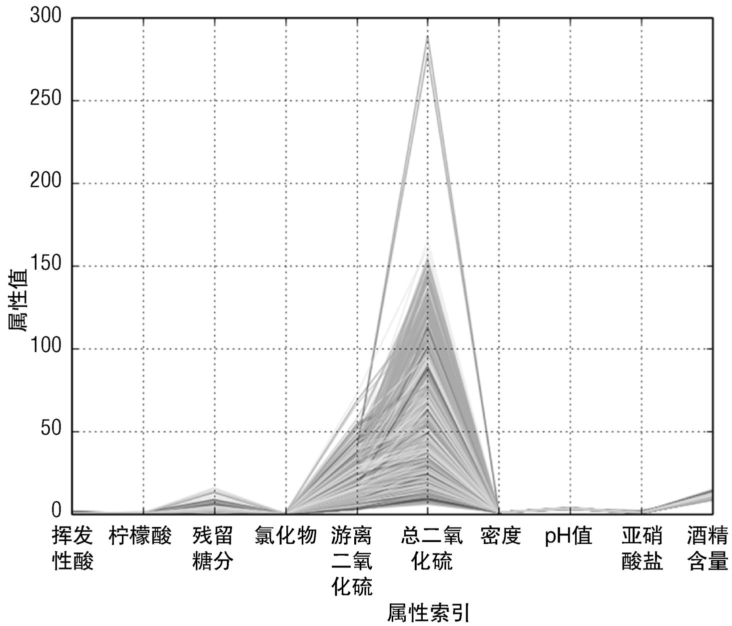


图 2-17 红酒数据的平行坐标图

为了克服这个问题，代码清单 2-15 对红酒数据进行了归一化，然后重画了平行坐标图。图 2-18 为归一化之后的平行坐标图。

代码清单 2-15 红酒数据的平行坐标图 – wineParallelPLot.Py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot
from math import exp

target_url = "http://archive.ics.uci.edu/ml/machine-learning-databases/
wine-quality/winequality-red.csv"
wine = pd.read_csv(target_url,header=0, sep=";")

#generate statistical summaries
```

```
summary = wine.describe()
nrows = len(wine.index)
tasteCol = len(summary.columns)
meanTaste = summary.iloc[1,tasteCol - 1]
sdTaste = summary.iloc[2,tasteCol - 1]
nDataCol = len(wine.columns) -1

for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = wine.iloc[i,1:nDataCol]
    normTarget = (wine.iloc[i,nDataCol] - meanTaste)/sdTaste
    labelColor = 1.0/(1.0 + exp(-normTarget))
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()

wineNormalized = wine
ncols = len(wineNormalized.columns)

for i in range(ncols):
    mean = summary.iloc[1, i]
    sd = summary.iloc[2, i]
    wineNormalized.iloc[:,i:(i + 1)] =
    (wineNormalized.iloc[:,i:(i + 1)] - mean) / sd

#Try again with normalized values
for i in range(nrows):
    #plot rows of data as if they were series data
    dataRow = wineNormalized.iloc[i,1:nDataCol]
    normTarget = wineNormalized.iloc[i,nDataCol]
    labelColor = 1.0/(1.0 + exp(-normTarget))
    dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

plot.xlabel("Attribute Index")
plot.ylabel(("Attribute Values"))
plot.show()
```

归一化红酒数据的平行坐标图可以更方便地观察出目标与哪些属性相关。图 2-18 展示了属性间清晰的相关性。在图的最右边，深蓝线（高口感评分值）聚集在酒精含量属性的高值区域；但是图的最左边，深红线（低口感评分值）聚集在挥发性酸属性的高值区域。这些都是最明显的相关属性。在第 5 章和第 7 章的预测模型中将会对属性基于对预测所做的贡献进行评分，我们会看到预测模型是如何支撑上述这些观察结果的。

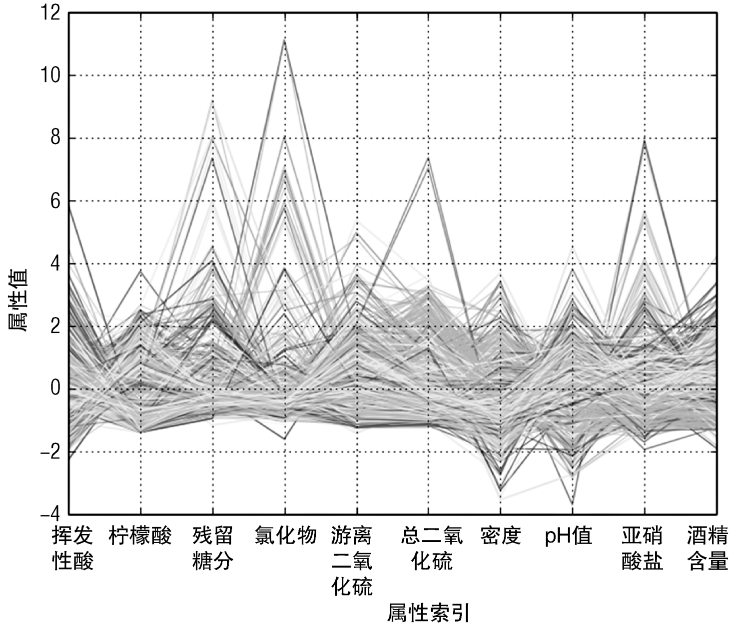


图 2-18 归一化红酒数据的平行坐标图

图 2-19 为属性之间、属性与目标之间的关联热图。在这个热图中，暖色对应强相关（颜色标尺的选择与平行坐标图中的正好相反）。红酒数据的关联热图显示口感评分值（最后一列）与酒精含量（倒数第二列）高度正相关，但是与其他几个属性（包括挥发性酸等）高度负相关。

分析红酒数据所用的工具在前面都已经介绍和使用过。红酒数据集展示了这些工具可以揭示的信息。平行坐标图和关联热图都说明酒精含量高则口感评分值高，然而挥发性酸高则口感评分值低。在第 5、第 7 章可以看到，预测模型中的一部分工作就是研究各种属性对预测的重要性。红酒数据集就是一个很好的例子，展示了如何通过探究数据来知晓向从哪个方向努力来构建预测模型以及如何评价预测模型。下节将探究多类别分类问题的数据集。

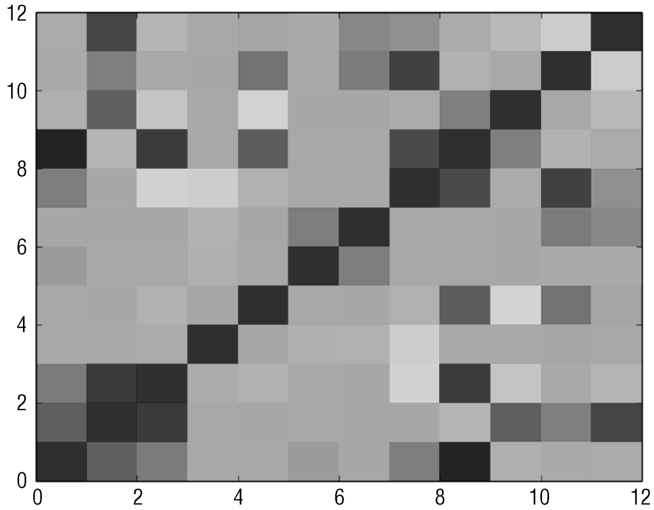


图 2-19 红酒数据的关联热图

2.6 多类别分类问题：它属于哪种玻璃

多类别分类问题与二元分类问题类似，不同之处在于它有几个离散的输出，而不是只有两个。回顾探测未爆炸的水雷的问题，它的输出只有两种可能性：声纳探测的物体是岩石或者水雷。而红酒口感评分问题根据其化学成分会产生几个可能的输出（其口感评分值是从 3 分到 8 分）。但是对于红酒口感评分问题，口感评分值存在有序的关系。打 5 分的红酒要好于打 3 分的，但是要劣于打 8 分的。对于多类别分类问题，输出结果是不存在这种有序关系的。

此节将根据玻璃的化学成分来判断玻璃的类型，目标是确定玻璃的用途。玻璃的用途包括建筑房间用玻璃、车辆上的玻璃、玻璃容器等。确定玻璃的用途类型是为了鉴证。例如在一个车祸或犯罪现场，会有玻璃的碎片，确定这些玻璃碎片的用途、来源，有助于确定谁是过错方或者谁是罪犯。代码清单 2-16 为生成玻璃数据集的统计信息的代码。图 2-20 为归一化玻璃数据的箱线图，箱线图显示有相当数量的异常点。

代码清单 2-16 玻璃数据集的统计信息 -glassSummary.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot
```

```
target_url = ("https://archive.ics.uci.edu/ml/machine-"
              "learning-databases/glass/glass.data")

glass = pd.read_csv(target_url,header=None, prefix="V")
glass.columns = ['Id', 'RI', 'Na', 'Mg', 'Al', 'Si',
                 'K', 'Ca', 'Ba', 'Fe', 'Type']

print(glass.head())

#generate statistical summaries
summary = glass.describe()
print(summary)
ncol1 = len(glass.columns)

glassNormalized = glass.iloc[:, 1:ncol1]
ncol2 = len(glassNormalized.columns)
summary2 = glassNormalized.describe()

for i in range(ncol2):
    mean = summary2.iloc[1, i]
    sd = summary2.iloc[2, i]

glassNormalized.iloc[:,i:(i + 1)] = \
       (glassNormalized.iloc[:,i:(i + 1)] - mean) / sd

array = glassNormalized.values
boxplot(array)
plot.xlabel("Attribute Index")
plot.ylabel(("Quartile Ranges - Normalized "))
show()

Output: [filename - ]
print(glass.head())

 Id      RI    Na   Mg   Al    Si    K   Ca Ba Fe Type
0 1 1.52101 13.64 4.49 1.10 71.78 0.06 8.75  0  0    1
1 2 1.51761 13.89 3.60 1.36 72.73 0.48 7.83  0  0    1
2 3 1.51618 13.53 3.55 1.54 72.99 0.39 7.78  0  0    1
3 4 1.51766 13.21 3.69 1.29 72.61 0.57 8.22  0  0    1
```

```
4 5 1.51742 13.27 3.62 1.24 73.08 0.55 8.07  0  0    1


print(summary) - Abridged
               Id          RI          Na          Mg          Al
count 214.000000 214.000000 214.000000 214.000000 214.000000
mean  107.500000   1.518365  13.407850   2.684533   1.444907
std    61.920648   0.003037   0.816604   1.442408   0.499270
min     1.000000   1.511150  10.730000   0.000000   0.290000
25%    54.250000   1.516523  12.907500   2.115000   1.190000
50%   107.500000   1.517680  13.300000   3.480000   1.360000
75%   160.750000   1.519157  13.825000   3.600000   1.630000
max   214.000000   1.533930  17.380000   4.490000   3.500000
                K          Ca          Ba          Fe        Type
count 214.000000 214.000000 214.000000 214.000000 214.000000
mean    0.497056   8.956963   0.175047   0.057009   2.780374
std     0.652192   1.423153   0.497219   0.097439   2.103739
min     0.000000   5.430000   0.000000   0.000000   1.000000
25%     0.122500   8.240000   0.000000   0.000000   1.000000
50%     0.555000   8.600000   0.000000   0.000000   2.000000
75%     0.610000   9.172500   0.000000   0.100000   3.000000
max     6.210000  16.190000   3.15000  0 0.510000   7.000000
```

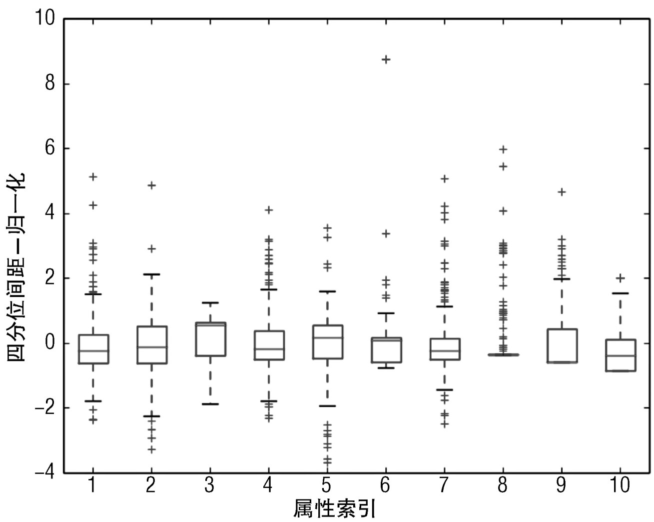


图 2-20 玻璃数据的箱线图

玻璃数据的箱线图显示有相当数量的异常点，至少与前面的例子相比，异常点数量上是比较多的。玻璃数据集有几个因素可能会导致出现异常点。首先这是一个分类问题，在属性值和类别之间不需要存在任何连续性，也就是说不应期望在各种类别之间，属性值是相互接近的、近似的。另外一个玻璃数据比较独特的地方是它的数据是非平衡的。成员最多的类有 76 个样本，而成员最小的类只有 9 个样本。统计时，平均值可能是由成员最多的那个类的属性值决定，因此不能期望其他的类别也有相似的属性值。采取激进的方法来区分类别可能会达到较好的结果，但这也意味着预测模型需要跟踪不同类别之间复杂的边界。在第 3 章可以了解到，如果给定足够多的数据，集成方法可以比惩罚线性回归方法产生更复杂的决策边界。而在第 5、第 7 章可以看到哪种方法可以获得更好的效果。

平行坐标图可能对此数据集揭示的信息更多。图 2-21 为其平行坐标图。数据根据输出类别用不同的颜色标记。有些类别区分度很好。例如，深蓝色的线聚集度很好，在某些属性上与其他类别的区分度也很好。深蓝的线在某些属性上经常处于数据的边缘，也就是说，是这些属性上的异常点。浅蓝的线在某些属性上也与深蓝的线一样，处于边缘地带，但是数量上要比深蓝的少，而且两者都在边缘地带时的所属的属性也不尽相同。棕色的线聚集性也很好，但其取值基本上在中心附近。

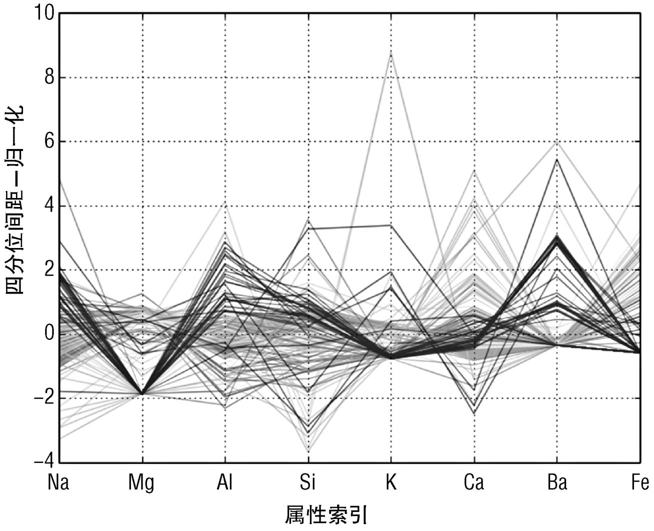


图 2-21 玻璃数据的平行坐标图

代码清单 2-17 为产生玻璃数据的平行坐标图的代码。针对岩石 vs. 水雷问题，平行坐标图的线用 2 种颜色代表了 2 种目标类别。在回归问题（红酒口感评分、鲍鱼预测年龄），标签（目标类别）取实数值，平行坐标图的线取一系列不同的颜色。在多类别分类问题中，

每种颜色代表一种类别，共有 6 种类别，6 种颜色。标签是 1 ～ 7，没有 4。颜色的选择与回归问题中的方式类似：将目标类别（标签）除以其最大值，然后再基于此数值选择颜色。图 2-22 为玻璃数据的关联热图。关联热图显示了属性之间绝大多数是弱相关的，说明属性之间绝大多数是相互独立的,这是件好事情。标签（目标类别）没有出现在热图中，因为目标（类别）只取几个离散值中的一个。不包括目标类别无疑减少了关联热图所能揭示的信息。

代码清单 2-17　玻璃数据的平行坐标图 -glassParallelPlot.py

```
__author__ = 'mike_bowles'
import pandas as pd
from pandas import DataFrame
from pylab import *
import matplotlib.pyplot as plot

target_url = ("https://archive.ics.uci.edu/ml/machine-"
              "learning-databases/glass/glass.data")

glass = pd.read_csv(target_url,header=None, prefix="V")
glass.columns = ['Id', 'RI', 'Na', 'Mg', 'Al', 'Si',
                 'K', 'Ca', 'Ba', 'Fe', 'Type']

glassNormalized = glass
ncols = len(glassNormalized.columns)
nrows = len(glassNormalized.index)
summary = glassNormalized.describe()
nDataCol = ncols - 1

#normalize except for labels
for i in range(ncols - 1):
    mean = summary.iloc[1, i]
    sd = summary.iloc[2, i]

glassNormalized.iloc[:,i:(i + 1)] = \
        (glassNormalized.iloc[:,i:(i + 1)] - mean) / sd

#Plot Parallel Coordinate Graph with normalized values
for i in range(nrows):
```

```
        #plot rows of data as if they were series data
        dataRow = glassNormalized.iloc[i,1:nDataCol]
        labelColor = glassNormalized.iloc[i,nDataCol]/7.0
        dataRow.plot(color=plot.cm.RdYlBu(labelColor), alpha=0.5)

    plot.xlabel("Attribute Index")
    plot.ylabel(("Attribute Values"))
    plot.show()
```

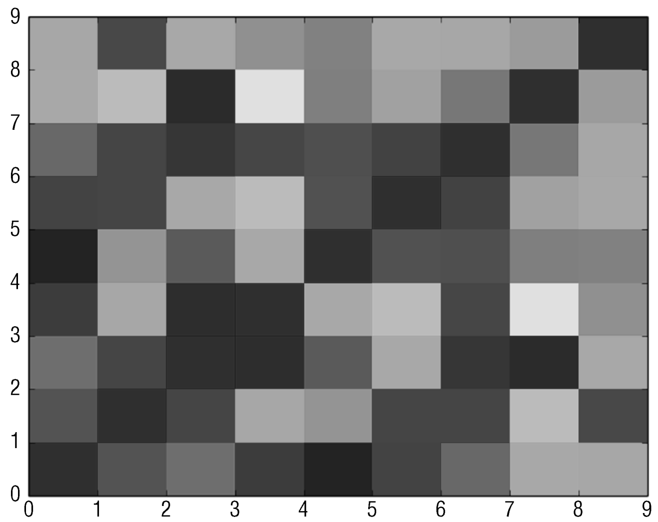


图 2-22 玻璃数据的关联热图

对玻璃数据的研究揭示了一个有趣的问题。具体地说，箱线图以及平行坐标图暗示了如果给定足够多的数据，采用集成方法是一个很好的选择。一系列的属性用来区分一个类别,明显类别之间会有复杂的边界。哪种算法会产生最佳的预测性能还有待进一步观察。本章学习的分析数据的方法已圆满完成了任务。它们可以帮助加深对问题的理解，通过各种权衡后可以更好地预判哪种算法可以获得较好的性能。

小结

本章介绍了用于探究新数据集的一些工具，接下来就是如何建立预测模型。这些工具从简单地获取数据集的规模开始，包括确定数据集属性和目标的类型等。这些关于数据集的基本情况会对数据集的预处理、预测模型的训练提供帮助。本章还包括一些统计概念，

帮助加深对数据的理解。这些概念包括：简单的统计信息（均值、标准差、分位数）、二阶统计信息，如属性间的相关性、属性与目标间的相关性。当目标是二值时，计算属性与目标相关性的方法与目标是实数（回归问题）时有所不同。本章也介绍了可视化技巧：利用分位数图来显示异常点；利用平行坐标图来显示属性和目标之间的相关性。上述方法和技巧都可以应用到本书后续的内容，用来验证算法以及算法之间的对比。

参考文献

Gorman, R. P., and Sejnowski, T. J.（1988）. UCI Machine Learning Repository.https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+%28Sonar,+Mines+vs.+Rocks%29. Irvine, CA: University of California, School of Information and Computer Science.

第 3 章

预测模型的构建：平衡性能、复杂性以及大数据

本章讨论影响机器学习模型效果的因素，并且给出了针对不同类型问题的性能定义。比如在电子商务应用方面，好的性能意味着返回正确的搜索结果或者展示网站访问者可能会点击的广告。对于基因问题，好的性能意味着发现与遗传现象相关的基因。本章将描述针对不同问题的相关性能指标。

选择并拟合一个预测算法的最终目标是获得最佳可能的效果。能够达到的性能取决于 3 方面因素：问题的复杂性、模型算法的复杂性以及可用的数据丰富程度。本章将通过一些可视化的例子来展示问题与模型复杂度的关系，并给出一些在设计和部署上的经验方法。

3.1　基本问题：理解函数逼近

本书涵盖的算法解决一类特定的预测问题。这类预测问题包括两种变量：

(1) 第一种变量是尝试要预测的变量（比如网站的访问者是否会点击广告这样的变量）。

(2) 第二种变量是用来进行预测的变量（比如网站访问者在人口统计方面的背景或者访问者在网站留下的历史访问行为这样的变量）。

这类问题称作函数逼近问题，因为目标是构建以第二类变量作为输入的函数来预测第一类变量。

在一个函数逼近问题中，模型设计者一般是从带有标记的历史样本集合开始研究。例如，网络日志文件会表明访问者是否会点击呈现的广告。数据科学家接着要从中找到可以用于构建预测模型的特征。例如，为了预测网站访问者是否会点击广告，数据科学家可能尝试使用访问者在看到广告前浏览过的其他页面。如果用户在网站注册过，那么历史购买数据或网页浏览记录都可用于预测。

要预测的变量一般有多种正式名称，如目标、标签、结果。用于构建预测的输入变量也有多种名称，如预测因子、回归因子、特征以及属性。这些词在下文中可能会换着使用，在实际应用中也不互相区分。决定使用哪种属性进行预测被称作特征工程。数据清洗以及特征工程占据了数据科学家 80% ～ 90% 的时间。

特征工程一般需要通过一个由人工参与的、迭代的过程来完成特征选择，决定可能最优的特征，并且尝试不同的特征组合。本书涵盖的算法将为每个属性赋一个重要度得分。这些得分表明了属性在构建预测时的相对重要性，从而帮助加速特征工程。

3.1.1 使用训练数据

数据科学家往往是从一个训练集开始进行算法开发。训练集包括结果以及由数据科学家选择的特征集合。训练集包括 2 类数据：

- 要预测的结果
- 用于预测的特征

表 3-1 展示了训练集中的一个样本。最左侧一列为实际结果（网站访问者是否点击了链接）；后两列是预测访问者是否会在将来点击链接的特征。

表 3-1 样例训练集

结果：是否点击链接	特征 1：用户性别	特征 2：用户在网站上的花销	特征 3：年龄
是	男	0	25
否	女	250	32
是	女	12	17

预测因子（即特征、属性等）可以使用矩阵的形式来表示（见公式 3-1）。本书约定使用的记号如下所示。预测因子构成的表使用 X 来表示。

$$X = \begin{matrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \ddots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{matrix}$$

公式 3-1 预测因子（特征）集合的表示符号

回到表 3-1 中的数据集，x_{11} 对应 M（用户性别），x_{12} 对应 0.00（网站上的花费），x_{21} 对应 F（性别），等等。

有时使用单个符号来指代一个特定样本的所有属性会很方便。比如 x_i（使用单个索引）对应 X 的第 i 行。对于表 3-1 中的数据集，x_2 是指包含值 F,250,32 的行向量。

严格来讲，X 不是矩阵，因为预测因子的数据类型可能不完全相同（一个合理矩阵包含的变量需要是相同类型，然而预测因子是不同类型）。以广告点击预测为例，预测因子可能包含网站访问者的人口背景数据，如婚姻状态、年收入等。年收入是个实数值，婚姻状态是一个类别变量。这意味着婚姻状态并不能进行数值运算，如相加、相乘，并且单身、

结婚、离异也不存在大小顺序关系。对于 X 中的列，数据类型相同，但对不同的列，数据类型可能不同。

对于婚姻状态、性别或者居住状态这类属性有统一的名称，如因子属性或者类别属性。类似于年龄或者收入的属性称作数值型或者实数型属性。这两类属性区别很重要，因为一些算法可能只能处理其中的一类属性。例如，线性方法包括本书提到的算法，需要使用数值属性（第 4 章涵盖了线性方法，介绍了将类别属性转换为数值属性的方法，以便将线性方法应用到包含类别属性的问题中）。

X 每一行对应的预测目标值排列起来可以用列向量 Y 来表示（参见公式 3-2），表示如下：

$$Y = \begin{matrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{matrix}$$

公式 3-2　目标向量的表示符号

预测目标 y_i 对应于输入 x_i- 即 X 的第 i 行中的预测结果。参照表 3-1 中的数据，y_1 表示“是”，y_2 表示“否”。

预测目标也可能有多种数据类型。一种是实数类型，如预测消费者会花多少钱。当预测目标是实数时,这类问题称作回归问题。线性回归是指使用线性方法来解决回归问题(本书同时涵盖了线性以及非线性方法)。

如果预测目标只包含 2 个值，如表 3-1 所示，那么对应问题称作 2 分类问题。预测用户是否会点击广告是一个 2 分类问题。如果预测目标包含多个不同的离散值，那么该问题称作多分类问题。在有多个广告的情况下,预测用户会点击哪个广告就是一个多分类问题。

回归问题的目标是寻找一个预测函数 pred()，该函数使用属性来预测输出结果（参照公式 3-3）。

$$y_t \sim pred(x_t)$$

公式 3-3　构建预测的基本公式

函数 pred() 使用属性 x_i 来预测 y_i。本书将介绍目前几种构建 pred ( ) 函数的最佳方法。

3.1.2　评估预测模型的性能

好的性能意味着使用属性 x_i 来生成一个接近真实 y_i 的预测，这种“接近”对不同问题含义不同。对于回归问题，y_i 是一个实数，性能使用均方误差（MSE）或者平均绝对误差（MAE）来度量（参见公式 3-4）。

$$Mean\ squared\ error = \left(\frac{1}{m}\right)\sum_{i=1}^{m}(y_i - pred(x_i))^2$$

公式 3-4 一个回归问题的性能度量

因为在一个回归问题中，预测目标（y_i）以及预测函数输出 pred（x_i）都是实数，所以通过它们的数值差异来描述错误是合理的。MSE 的计算公式 3-4 对误差求平方，然后平均来生成对错误的整体度量。MAE 对误差的绝对值求平均（参照公式 3-5）而不是对误差平方求平均。

$$Mean\ absolute\ error = \left(\frac{1}{m}\right)\sum_{i=1}^{m}|\ y_i - pred(x_i)\ |$$

公式 3-5 回归性能的另一种度量指标

如果问题是一个分类问题，那么需要使用不同的性能指标。最常用的性能指标是误分类率，即计算 pred() 函数预测错误的样本比例。3.3.1“不同类型问题的性能评价指标”介绍了如何计算误分类率。

对于预测函数 pred()，需要评估其在新样本（未见过的）上的错误程度。算法在新数据上（这些数据在生成预测函数 pred() 时并没有使用）的表现如何？本章将介绍用于评估算法在新数据上性能的最佳方法。

本节介绍了预测问题的基本类型，阐明构建预测模型为何等同于构建将属性（或者特征）映射为输出的函数。还介绍了评估预测错误的方法。以上步骤涉及若干难点，下面内容将对这些难点一一描述并且解决，介绍如何在问题及数据限制的情况下达到最佳效果。

3.2 影响算法选择及性能的因素——复杂度以及数据

有几个因素影响预测算法的整体性能。这些因素包括问题的复杂度、模型复杂度以及可用的训练数据量。下面将介绍这些因素是如何一起来影响预测性能的。

3.2.1 简单问题和复杂问题的对比

前面小节的内容介绍了性能评估的几种方式，强调模型在新数据上的性能表现更为重要。设计预测模型的目标是在新样本上（如网站新用户）预测准确。应用数据科学家需要对算法性能进行评估，从而为客户提供合理的期望并且对算法进行比较。对模型进行评估的最佳经验是从训练数据集中预留部分数据。

这些预留的数据包含类别标签，从而可以与模型生成的预测结果进行比对。统计学家将这种比对结果称作样本外误差，因为计算误差的样本并没有在训练中用到（3.3 节会深入讨论该过程的运行机制）。记住只有在新样本上计算得到的性能才能算是模型的性能。

影响性能的一个因素是问题复杂度。图 3-1 展示了一个简单的 2 分类问题，输入包含 2 个维度。有 2 组数据点：深色的点和浅色的点。深色数据点是随机从二维高斯分布中抽样，中心在（1,0），方法是 2 个方向的单位方差。浅色的点也是从高斯分布中抽样，有相同的方差，但是中心在（0,1）。问题的输入属性对应图 3-1 中的 2 个坐标轴：x_1 以及 x_2。分类任务对应于在 x_1,x_2 的二维平面上画一条分界线来分离浅色点以及深色点。在这种情况下，最好的解决方案是在图中画一个 45 度线，即 x_1=x_2 的直线。从概率意义上讲，这是最好的分类器。因为一条直线尽可能地分离了浅色点以及深色点。本书要介绍的线性方法将会很好地解决该问题。

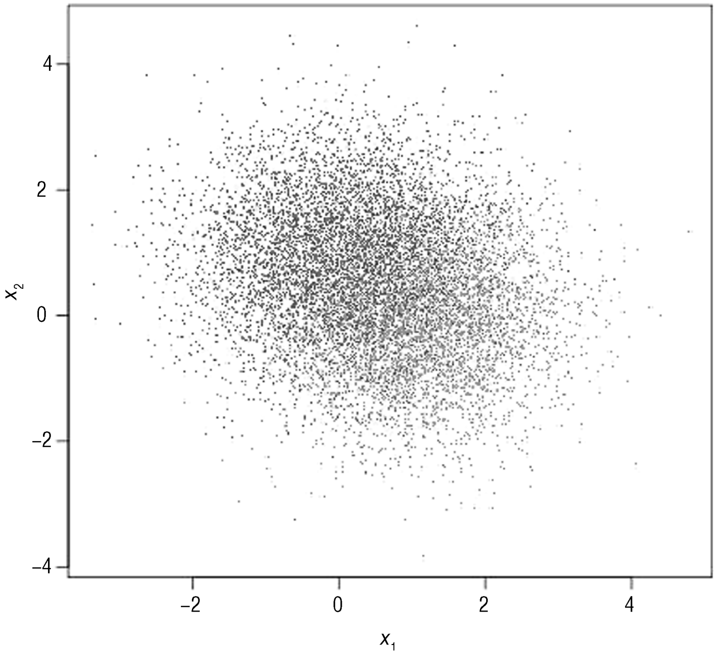


图 3-1　一个简单的分类问题

图 3-2 展示了一个更加复杂的问题。图中的点通过随机抽样来生成。与图 3-1 中的随机抽样不同，图 3-2 中的点从多个浅色点以及深色点的分布中抽样，这种生成数据的模型被称作混合模型。不过这两个问题的总体目标是相同的：在 x_1，x_2 的平面上画一条边界来分离浅色点与深色点。在图 3-2 中，很明显一条线性边界不能将点分离，一条曲线也不能分离。第 6 章中提到的集成方法可以很好地解决该问题。

然而决策边界的复杂性并不是唯一决定使用线性方法还是非线性方法的因素。另一个重要因素是数据集的大小。图 3-3 展示了数据集大小对性能的影响。图 3-3 中的点从图 3-2 中抽样得到，抽样比例为 1%。

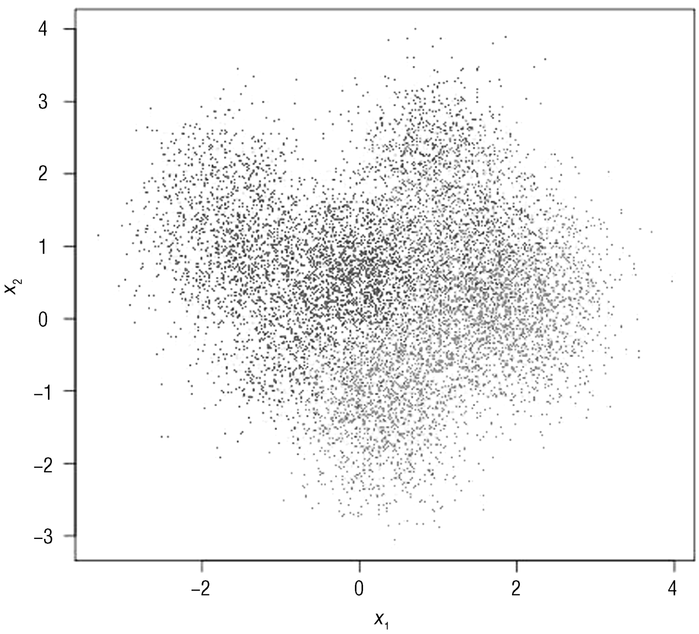


图 3-2 一个复杂的分类问题

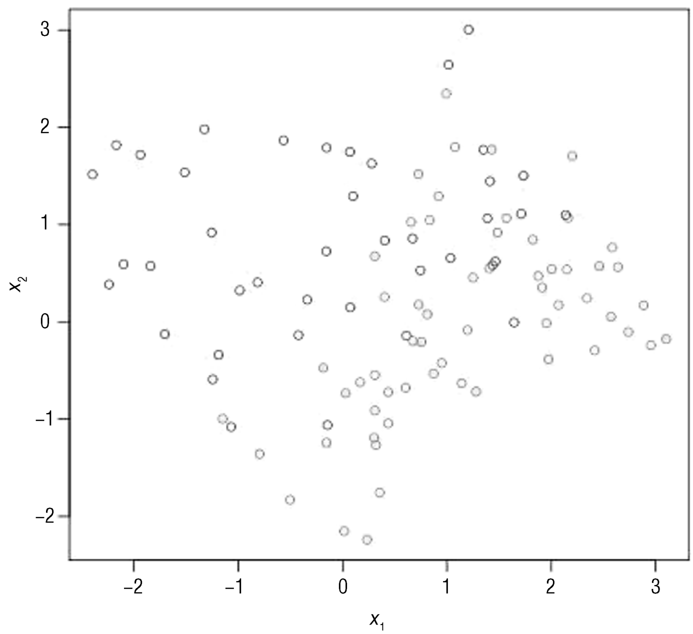


图 3-3 没有太多数据的复杂分类问题

图 3-2 包含充足的数据，人们可以通过视觉来确定边界，这些边界很好地区分了浅色

点以及深色点。如果没有这么多数据点，不同类别就很难在视觉上进行区分，此时，相比非线性模型，一个线性模型可以给出相同甚至更好的性能。如果数据少的话，边界很难通过视觉确定，所以也更难进行计算。本例通过图像展示了数据量对于分类的重要性。如果问题很复杂（如对不同购物者提供个性化的服务），一个拥有大量数据的复杂模型可以生成精确结果。然而，如果真实模型不复杂（见图 3-1）或者没有足够多的数据（见图 3-3），一个线性模型可能是最好的答案。

3.2.2 一个简单模型与复杂模型的对比

前面介绍了简单问题和复杂问题在视觉上的差异。本节将描述能够解决这些问题的不同模型是如何工作的以及它们之间的差异。直观上，一个复杂的模型应该解决复杂的问题，但是上一节的可视化的例子表明在数据有限情况下，对于复杂问题，简单模型可能要好于复杂模型。

另一个重要概念是现代机器学习算法往往生成模型族，不只是单个模型。本书提到的每个算法可以生成成百上千个不同的模型。一般来讲，第 6 章提到的集成方法可以产生比第 4 章中提到的线性方法更多的复杂模型，但是两种方法都可以生成不同复杂度的多个模型（通过第 4 章和第 6 章对线性方法和集成方法的深入讨论，该结论会更加清晰）。

图 3-4 为上一章中用于拟合简单问题的线性模型。图 3-4 中的线性模型使用 glmnet 算法（第 4 章会提到）生成。拟合这些数据的线性模型将数据粗暴地分为 2 份。图 3-4 中的分界线对应于公式 3-6。

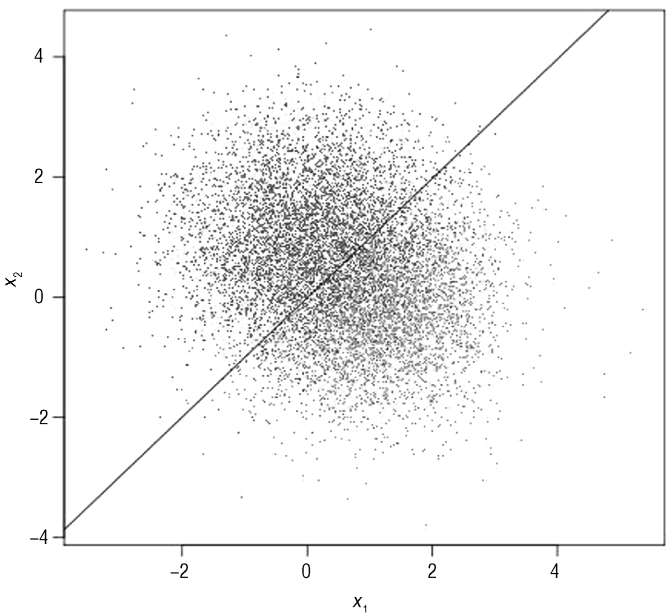


图 3-4 拟合简单数据的线性模型

$$x_2 = -0.01 + 0.99x_1$$

公式 3-6 适用于简单问题的线性模型

该分界线非常接近于 $x_2=x_1$ 的分界线，从概率意义上讲，$x_2=x_1$ 是最佳可能的分界线。该边界从视觉角度讲也是合理的。针对该简单问题拟合一个更加复杂的模型不能进一步提升预测效果。

一个拥有更多决策边界的复杂的问题为复杂模型提供了超越简单线性模型的机会。图 3-5 为拟合数据后的线性模型，线性模型效果表现不尽人意，表明该问题实际需要非线性的决策边界。在这种情况下，线性模型会将浅色点错分为深色点，反之亦然。

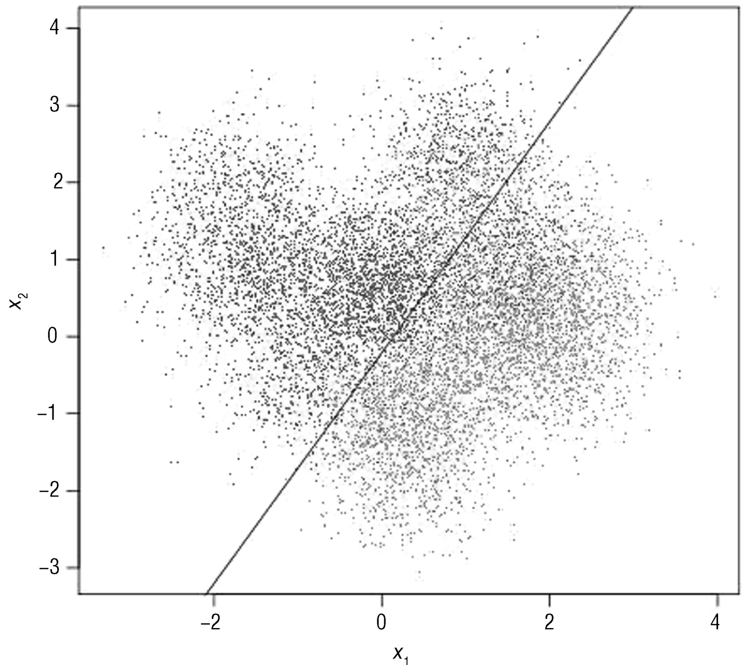


图 3-5 拟合复杂数据的线性模型

图 3-6 展示了复杂模型如何更好地处理复杂数据。用于生成该决策边界的模型是一个包含 1 000 棵二分类决策树的集成模型，该模型通过梯度提升算法训练得到（第 6 章会详细介绍梯度提升决策树）。非线性决策边界用于更好地划分深色和浅色区域。

虽然倾向于得到如下结论：即最好的办法是用复杂模型解决复杂问题，用简单模型解决简单问题，但是必须考虑问题的另一维度。正如在前面章节中提到的，必须考虑数据规模，图 3-7 以及图 3-8 为从复杂问题抽样的 1% 的数据。图 3-7 为一个拟合数据的线性模型，图 3-8 为一个拟合数据的集成模型。统计误分类的点数，目前数据集中共有 100 个点。图 3-7

中的线性模型误分类了 11 个点，错误率为 11%。复杂模型误分类了 8 个点，错误率为 8%。它们的性能基本一样。

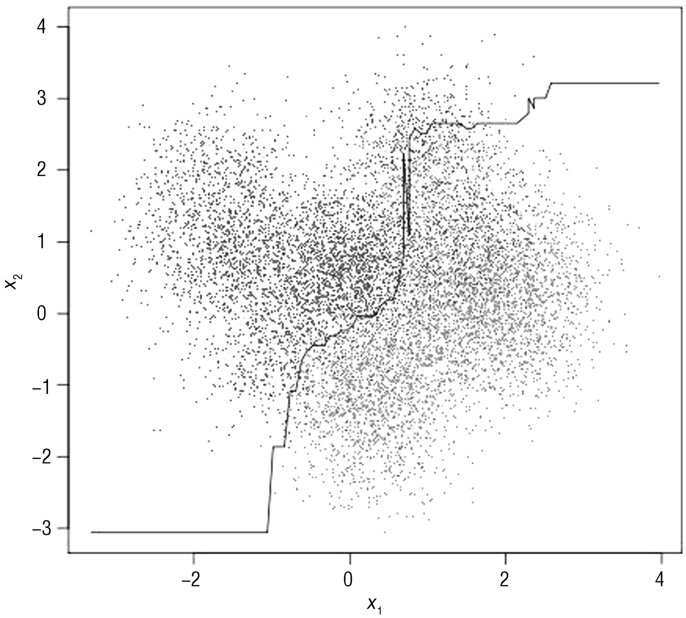


图 3-6 拟合复杂数据的集成模型

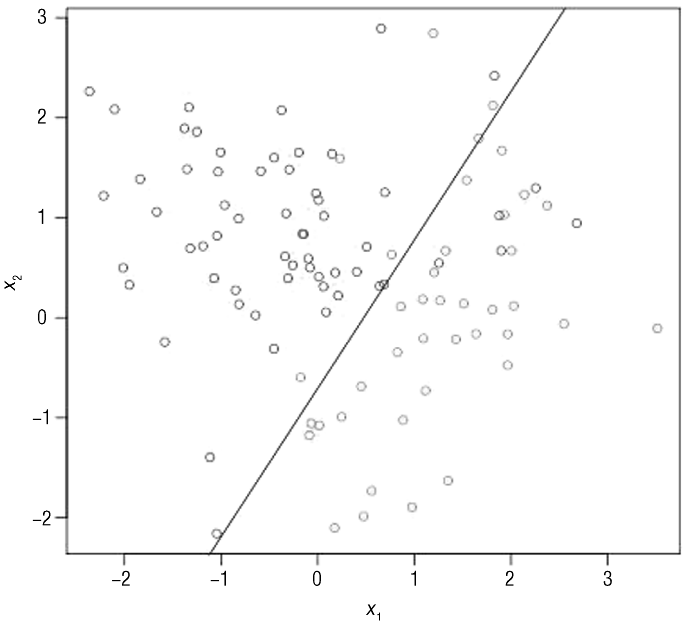


图 3-7 拟合少量复杂数据样本的线性模型

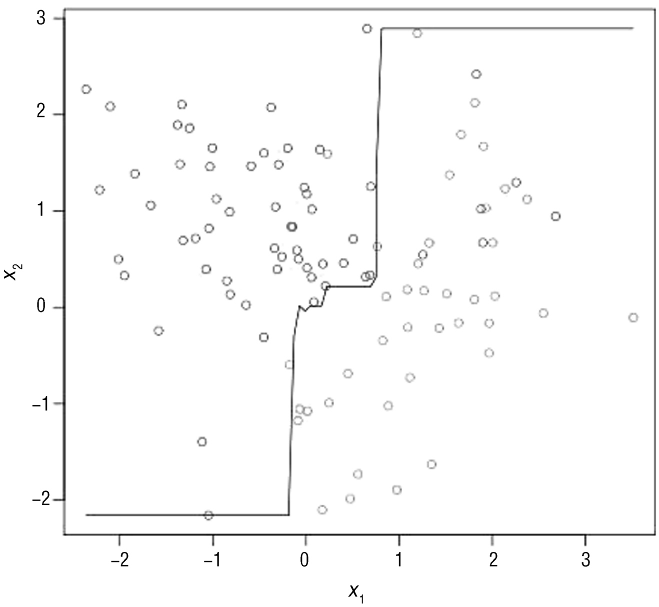


图 3-8 拟合少量复杂数据的集成模型

3.2.3 影响预测算法性能的因素

上面结果说明了大数据对于预测的重要性。对于复杂问题进行精确预测需要大量数据，但到底需要多少数据其实很难准确描述。数据的分布形状也很重要。

公式 3-1 将数据描绘为一个矩阵，该矩阵包含大量的行（高）和列（宽）。矩阵中的元素个数是行数与列数的乘积。当数据用于预测模型时，行数与列数的多少会对预测产生重要影响。添加一列意味着添加一个新的属性，添加一行意味着添加一条使用当前属性表示的新样本。为了理解添加一个新行与一个新列的不同，考虑一个线性模型，该线性模型使用来自公式 3-1 的属性以及来自公式 3-2 的标签。

假设模型的表示形式如下（参见公式 3-7）。

$$y_i \sim x_i * \beta$$
$$= x_{i1} * \beta 1 + x_{i2} * \beta 2 + \cdots + x_i m * \beta m$$

公式 3-7 属性和输出的线性关系

这里，x_i 是一行属性，β 是一列要学习的系数。添加一列对应于新添加一个要学习的参数。这里系数个数也称作自由度。增加额外的自由度会使模型变复杂。前面例子表明复杂模型需要更多的数据。这种情况可以通过行数与列数比例来考虑，即长宽比。

生物数据集以及自然语言处理数据集一般是包含大量列的数据集，这些数据集虽然有

很多样本，但往往也不足以训练好一个复杂模型。在生物学里，基因数据集很容易就包含 10,000 ～ 50,000 个属性。即使通过成百上千次的单个实验（数据的行），基因数据也不足以训练一个复杂的集成模型。线性模型可以给出等价甚至更好的性能。

基因数据很昂贵。一次实验（数据行）就可能花费 $5,000 美元，整个数据集花费可能会达到 5,000 万美元。文本相对容易收集和存储，但属性个数可能要比基因数据中的属性个数更多。对于一些自然语言处理问题，属性是词，每一行对应一篇文档。属性矩阵中的每一个元素表示词在文档中的出现次数。列的数目对应于文档的词汇量大小。根据预处理情况（如移除常见的词，如 a、and 以及 of），最后的词汇量可能会从几千到数万。如果考虑 n-gram，文本的属性矩阵会更加庞大。n-gram 是相邻的 2 个、3 个或者 4 个词，这些词的位置足够紧密甚至可以构成短语。在这种情况下，线性模型相对于复杂的集成方法，可能会产生相同甚至更好的性能。

3.2.4 选择一个算法：线性或者非线性

刚刚看到的可视化例子说明了使用线性或者非线性模型应该做的权衡考虑。对于列比行多的数据集或者相对简单的问题，倾向于使用线性模型。对于行比列多很多的复杂问题，倾向于使用非线性模型。另一个考虑因素是训练时间。线性方法要比非线性方法训练时间短（当你读完第 4 章以及第 6 章并且实际操作过一些例子时，你会有更多的经验来选择方法）。

选择一个非线性模型（如集成方法）对应于训练大量不同复杂度的模型。例如，生成图 3-6 所示的决策边界的集成模型在训练时大约会生成 1000 个不同的模型。这些模型有不同的复杂度。一些模型会给出非常粗粒度的近似边界，这些边界如图 3-6 所示。之所以选择图 3-6 中的决策边界，是因为对应模型在预留数据集上的效果最好。以上过程对许多现代的机器学习算法同样适用。3.4.1 节会给出具体例子。

通过本节使用的数据集以及分类器进行可视化，读者能够直观地看到影响预测模型性能的因素。通常来讲，一般使用数值评价指标来度量性能而非依赖于图形展示。下一章将介绍基于数值指标的评价方法、使用评价指标要考虑的因素以及如何使用评价指标来评估部署模型的性能。

3.3 度量预测模型性能

本节介绍针对预测模型进行性能度量的两个方面。第一个方面是对不同问题使用不同指标（如对回归问题使用 MSE、对分类问题使用误分类率）。在相关文献（以及机器学习竞赛）中，也可以看到使用 ROC 曲线以及曲线下面积（AUC）的评价指标。除了评价，这些指标对于性能优化也很重要。

第二个方面是在预留样本上进行错误估计的技术。预留样本错误是指在新数据上的错误率。如何利用上述技术来进行算法比较和模型选择是实践的重要问题。本章后续会对技术进行详细讨论，并且之后使用的例子也遵照该过程。

3.3.1 不同类型问题的性能评价指标

回归问题使用的性能指标相对比较直观。在回归问题中，真实目标以及预测值都是实数。错误很自然地被定义为目标值与预测值的差异。生成错误的统计摘要对性能比较以及问题诊断都非常有用。最常用的错误摘要是均方误差（MSE）以及平均绝对错误（MAE）。MSE、MAE 以及根 MSE（也写作 RMSE，即 MSE 的平方根）的比较如代码清单 3-1 所示。

代码清单 3-1 MSE、MAE 以及 RMSE- regressionErrorMeasures.py

```
__author__ = 'mike-bowles'


#here are some made-up numbers to start with
target = [1.5, 2.1, 3.3, -4.7, -2.3, 0.75]
prediction = [0.5, 1.5, 2.1, -2.2, 0.1, -0.5]
error = []
for i in range(len(target)):
    error.append(target[i] - prediction[i])

#print the errors
print("Errors ",)
print(error)
#ans: [1.0, 0.6000000000000009, 1.1999999999999997, -2.5,
#-2.3999999999999999, 1.25]


#calculate the squared errors and absolute value of errors
squaredError = []
absError = []
for val in error:
    squaredError.append(val*val)
    absError.append(abs(val))

#print squared errors and absolute value of errors
print("Squared Error")
```

```
print(squaredError)
#ans: [1.0, 0.3600000000000001, 1.4399999999999993, 6.25,
#5.7599999999999998, 1.5625]
print("Absolute Value of Error")
print(absError)
#ans: [1.0, 0.6000000000000009, 1.1999999999999997, 2.5,
#2.3999999999999999, 1.25]

#calculate and print mean squared error MSE
print("MSE = ", sum(squaredError)/len(squaredError))
#ans: 2.72875

from math import sqrt
#calculate and print square root of MSE (RMSE)
print("RMSE = ", sqrt(sum(squaredError)/len(squaredError)))
#ans: 1.65189285367

#calculate and print mean absolute error MAE
print("MAE = ", sum(absError)/len(absError))
#ans: 1.49166666667

#compare MSE to target variance
targetDeviation = []
targetMean = sum(target)/len(target)
for val in target:
    targetDeviation.append((val - targetMean)*(val - targetMean))
#print the target variance
print("Target Variance = ", sum(targetDeviation)/len(targetDeviation))
#ans: 7.5703472222222219

#print the the target standard deviation (square root of variance)
print("Target Standard Deviation = ", sqrt(sum(targetDeviation)
   /len(targetDeviation)))
#ans: 2.7514263977475797
```

本例从一组构造的目标值与预测值开始。首先，通过简单相减来计算错误；然后给出了 MSE、MAE 以及 RMSE 的计算方法。注意到 MSE 在量级上与 MAE 及 RMSE 都明显不同。这是因为 MSE 是平方级别。从这个角度讲，RMSE 是一个可用性更好的指标。代码清单最后是方差的计算（与均值的均方误差）以及标准差计算（方差的开方）。这些量与

预测错误指标 MSE 以及 RMSE 进行比较非常有意义。例如，如果预测错误的 MSE 同目标方差几乎相等（或者 RMSE 与目标标准差几乎相等），这说明预测算法效果并不好，通过简单对目标值求平均来替换预测算法就能达到几乎相同的效果。代码清单 3-1 中所示的预测错误 RMSE 大约是实际目标标准差的一半。这已经是一个相当不错的性能。

除了计算错误的摘要统计量以外，查看错误的分布直方图、长尾分布（使用分位数或者等分边界）以及正态分布程度等对于分析错误原因以及错误程度也非常有用。有时这些探索会对定位错误原因以及提升潜在性能带来启发。

分类问题需要不同对待。分类问题的评价方法一般围绕误分类率展开，即错分的样本所占的比例。例如，预测网站访问者是否会点击链接是一个分类问题。分类算法可以给出预测概率，而不是硬的决策（输出结果只有点击以及未点击）。本书讨论的算法都会输出概率。

下面展示为什么概率是有用的。如果点击或者不点击的预测结果以概率的方式给出，假设 80% 的情况下会点击（对应的 20% 的情况不点击），数据科学家可以选择 50% 作为一个阈值来决定是否呈现链接。在一些情况下，设置更高或者更低的阈值会给出更好的预测结果。

假设问题是欺诈检测（对于信用卡、自动票据交换（支票）、保险索赔等）。判断是否欺诈需要有电话座席员人工介入交易来放行。不同决策对应不同代价：如果产生了电话，就会有电话费以及客户响应的代价；如果没有电话，就存在被欺诈的代价。如果采取行动的代价相对于不采取行动的代价非常低，此时可以使用低的阈值来决定是否采取行动，从而欺诈代价就比较低，但同时会有更多的人工介入。

那么在什么时候应该介入客户提现，要求其必须致电信用卡服务中心来完成后续操作？当算法得到此笔交易存在欺诈的概率为 20%、50% 或者 80% 的概率时，开始介入交易？如果设置介入的阈值为 20%，你会更加频繁地介入，避免更多的欺诈交易，但同时也会激怒更多客户以及增加坐席代表工作负担。或许将阈值设高（如 80%）来容忍更多的欺诈是一个更好的策略。

对于这种情况，一般使用混淆矩阵（confusion matrix）或者列联表（contingency table）来安排可能的结果输出。图 3-9 为混淆矩阵的一个例子。混淆矩阵中的数字表示基于指定阈值进行决策所产生的性能值。图 3-9 中的混淆矩阵是基于特定的阈值对 135 个测试样例进行预测后的结果。矩阵中的 2 列代表可能的预测值，2 行表示每个样本可能取到的真实值。所以测试集中的每个样本可以被分配到表中 4 个单元中的一个。图 3-9 的 2 类对应于“点击”以及“未点击”，该分类结果用于选择是否呈现广告。2 类也可以对应于“欺诈”以及“非欺诈”，或者其他结果对，这取决于要解决的问题。

		预测类	
实际类		正例（点击）	负例（未点击）
	正（点击）	真正（TP）	假负（FN）
	负（未点击）	假正（FP）	真负（TN）

图 3-9 混淆矩阵的样例

左上角的单元格包含预测结果为点击并且预测标签与真实类别标签一致的样本。这些样本称作真正例，一般简写为 TP。左下部分的单元格对应预测是正的（点击），但是实际答案是负的（未点击）。这些样本称作假正例，简写为 FP。矩阵右侧的列包含预测结果为未点击的样本。右上角样本的正确答案是点击因此被称作假负例或者 FN。右下角的样本被预测为未点击，与正确答案一致，称作真负例或者 TN。

如果决策阈值改变会发生什么？考虑一种极端情况。将决策阈值设为 0.0，不论模型预测的概率是多少，结果都会被认为是点击。此时所有样本都会挤到左边一列。右边一列只有 0。TP 的数量会变为 17。FP 的数量会变为 118。如果对 FP 没有惩罚，TN 没有奖励，这种方案还说得下去，但如果假设样本都是点击的话，就不需要预测算法了。类似地，如果对 FN 没有惩罚，对 TP 没有奖励，阈值可以设为 1.0，这样所有的样本被分类为未点击。通过这些极端例子可以帮助我们理解决策阈值的作用，但对实际应用用处不大。下面我们将展示在岩石 - 水雷数据集上构建分类器的过程。

岩石 - 水雷数据集对应的问题是使用声纳数据来判断海底物体是岩石还是水雷（如果想全面讨论或者探究数据集，参照第 2 章）。代码清单 3-2 为在岩石 - 水雷数据集上训练简单分类器的 Python 代码。

代码清单 3-2 在岩石 - 水雷数据集上度量分类器性能

```
__author__ = 'mike-bowles'
#use scikit learn package to build classified on rocks-versus-mines data
#assess classifier performance

import urllib2
import numpy
import random
from sklearn import datasets, linear_model
from sklearn.metrics import roc_curve, auc
import pylab as pl
```

```
def confusionMatrix(predicted, actual, threshold):
    if len(predicted) != len(actual): return -1
    tp = 0.0
    fp = 0.0
    tn = 0.0
    fn = 0.0
    for i in range(len(actual)):
        if actual[i] > 0.5: #labels that are 1.0 (positive examples)
            if predicted[i] > threshold:
                tp += 1.0 #correctly predicted positive
            else:
                fn += 1.0 #incorrectly predicted negative
            else:         #labels that are 0.0 (negative examples)
            if predicted[i] < threshold:
                tn += 1.0 #correctly predicted negative
            else:
                fp += 1.0 #incorrectly predicted positive
    rtn = [tp, fn, fp, tn]
    return rtn

#read in the rocks versus mines data set from uci.edu data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []
labels = []
for line in data:
    #split on comma
    row = line.strip().split(",")
    #assign label 1.0 for "M" and 0.0 for "R"
    if(row[-1] == 'M'):
        labels.append(1.0)
    else:
        labels.append(0.0)
    #remove lable from row
    row.pop()
```

```
    #convert row to floats
    floatRow = [float(num) for num in row]
    xList.append(floatRow)

#divide attribute matrix and label vector into training(2/3 of data)
#and test sets (1/3 of data)
indices = range(len(xList))
xListTest = [xList[i] for i in indices if i%3 == 0 ]
xListTrain = [xList[i] for i in indices if i%3 != 0 ]
labelsTest = [labels[i] for i in indices if i%3 == 0]
labelsTrain = [labels[i] for i in indices if i%3 != 0]

#form list of list input into numpy arrays to match input class
#for scikit-learn linear model
xTrain = numpy.array(xListTrain); yTrain = numpy.array(labelsTrain)
xTest = numpy.array(xListTest); yTest = numpy.array(labelsTest)

#check shapes to see what they look like
print("Shape of xTrain array", xTrain.shape)
print("Shape of yTrain array", yTrain.shape)
print("Shape of xTest array", xTest.shape)
print("Shape of yTest array", yTest.shape)

#train linear regression model
rocksVMinesModel = linear_model.LinearRegression()
rocksVMinesModel.fit(xTrain,yTrain)

#generate predictions on in-sample error
trainingPredictions = rocksVMinesModel.predict(xTrain)
print("Some values predicted by model", trainingPredictions[0:5],
 trainingPredictions[-6:-1])

#generate confusion matrix for predictions on training set (in-sample
confusionMatTrain = confusionMatrix(trainingPredictions, yTrain, 0.5)
#pick threshold value and generate confusion matrix entries
tp = confusionMatTrain[0]; fn = confusionMatTrain[1]
fp = confusionMatTrain[2]; tn = confusionMatTrain[3]

print("tp = " + str(tp) + "\tfn = " + str(fn) + "\n" + "fp = " +
```

```
str(fp) + "\ttn = " + str(tn) + '\n')

#generate predictions on out-of-sample data
testPredictions = rocksVMinesModel.predict(xTest)

#generate confusion matrix from predictions on out-of-sample data
conMatTest = confusionMatrix(testPredictions, yTest, 0.5)
#pick threshold value and generate confusion matrix entries
tp = conMatTest[0]; fn = conMatTest[1]
fp = conMatTest[2]; tn = conMatTest[3]
print("tp = " + str(tp) + "\tfn = " + str(fn) + "\n" + "fp = " +
str(fp) + "\ttn = " + str(tn) + '\n')

#generate ROC curve for in-sample

fpr, tpr, thresholds = roc_curve(yTrain,trainingPredictions)
roc_auc = auc(fpr, tpr)
print( 'AUC for in-sample ROC curve: %f' % roc_auc)

# Plot ROC curve
pl.clf()
pl.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
pl.plot([0, 1], [0, 1], 'k-')
pl.xlim([0.0, 1.0])
pl.ylim([0.0, 1.0])
pl.xlabel('False Positive Rate')
pl.ylabel('True Positive Rate')
pl.title('In sample ROC rocks versus mines')
pl.legend(loc="lower right")
pl.show()

#generate ROC curve for out-of-sample
fpr, tpr, thresholds = roc_curve(yTest,testPredictions)
roc_auc = auc(fpr, tpr)
print( 'AUC for out-of-sample ROC curve: %f' % roc_auc)

# Plot ROC curve
pl.clf()
pl.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
```

```
pl.plot([0, 1], [0, 1], 'k-')
pl.xlim([0.0, 1.0])
pl.ylim([0.0, 1.0])
pl.xlabel('False Positive Rate')
pl.ylabel('True Positive Rate')
pl.title('Out-of-sample ROC rocks versus mines')
pl.legend(loc="lower right")
pl.show()
```

代码第一部分将 Irvine 数据集读入并将其解析为包含标签与属性的记录。下一步将数据划分为 2 个子集：测试集包含 1/3 的数据，训练集包含剩下 2/3 的数据。标注为 test 的数据集不能用于训练分类器，但会被保留用于评估训练得到的分类器性能。这一步用来模拟分类器在新数据样本上的行为。稍后本章将会讨论用于预留数据的方法，并在新数据集上对性能进行评估。

分类器的训练通过将标签 M（代表水雷）以及标签 R（代表岩石）转换为 2 个数值：1.0 对应于矿产，0.0 对应于岩石，然后使用最小二乘法来拟合一个线性模型。上面方法理解实现起来都非常简单，性能也接近于后续讨论的更加复杂的方法。代码清单 3-2 中的程序使用 scikit-learn 中的线性回归包来训练普通的最小均方模型。训练的模型用于在训练集和测试上生成预测。

代码会打印一些预测值的样例。线性回归模型产生的预测大部分集中在 0.0 到 1.0，然而也并非全部。这些预测不只是概率。仍然可以将它们与决策阈值进行比较来生成分类标签。函数 confusionMatrix() 生成了混淆矩阵，类似于图 3-9。该函数以预测值、对应的实际标签以及决策阈值作为输入。函数将预测值与决策阈值进行比较来决定为每个样本赋“正值”或者“负值”，预测值对应于混淆矩阵中的列。函数再根据样本的实际标签将样本放在对应的行中。

每个决策阈值的错误率都可以从混淆矩阵中计算得到。总的错误数为 FP 与 FN 的加和。样例代码分别在训练集以及测试集上计算混淆矩阵，并且打印出来。在训练集上误分类率为 8%，在测试集上误分类率为 26%。一般来讲，测试集上的性能要差于训练集上的性能。在测试集上的结果更能代表错误率。

当决策阈值改变的话，误分类率也会改变。表 3-2 显示随着决策阈值的变化，误分类率的变化情况。表中数字基于测试集计算得到。书中后续所有性能描述都是基于测试集的结果。如果错误基于训练集，书中会有提示：“警告：这些是在训练集上得到的错误”。如果目标是最小化误分类错误，那么最佳的决策阈值应该设为 0.25。

表 3-2 决策阈值对误分类率的影响

决 策 阈 值	误 分 类 率
0	28.6%
0.25	24.3%
0.5	25.7%
0.75	30.0%
1.0	38.6%

最佳决策阈值应该是能够最小化误分类率的值。然而，不同类错误对应的代价可能是不同的。例如，对于岩石 - 水雷预测问题，如果将岩石预测为水雷，可能会花费 \$100 请潜水员下水确认；如果将水雷预测为岩石，那么未爆炸的水雷不移除的话可能会导致 \$1,000 美元的人身财产损失。一个 FP 的样本代价为 100，一个 FN 的样本代价为 1,000。有了这样的假设，不同决策阈值生成的错误代价如表 3-3 所示。将水雷误分类为岩石（不对其进行处理可能会威胁到健康安全）的高代价会使最优决策阈值趋向于 0。这意味着会产生更多 FN，因为 FN 的代价不高。一项完整的分析应该包括移除水雷的代价以及随着移除带来的 1,000 美元的好处。如果这些数值已经知道（或者接近于合理近似），它们理应在计算阈值时考虑。

表 3-3 不同决策阈值的错误代价

决策阈值	假负例的代价	假正例的代价	总代价
0.0	1000	1900	2900
0.25	3000	1400	4400
0.5	9000	900	9900
0.75	18000	300	18300
1.00	26000	100	26100

注意到总的 FP 以及 FN 的相对代价取决于数据集中正例与负例的比例。岩石 - 水雷数据集有相同数量的正例与负例（岩石和水雷）。这些都是实验中的常用假设。实际遇到的正例数和负例数可能完全不同。在系统部署场景下，如果正负例数目不同，就可能要基于实际应用比例做一些调整。

数据科学家可能并没有关于正负样本误分的具体代价值，但仍然想使用除了误分类率外的其他刻画错误的方法。一种常见的指标被称作接收者操作曲线或者 ROC 曲线（http://en.wikipedia.org/wiki/Receiver_operating_characteristic）。

ROC 从其初始应用中继承了对应的名字：通过处理雷达信号来判断是否有敌机出现。ROC 曲线使用一个图来展示不同的列联表，图中绘制的是真正率（简写为 TPR）随假正率（FPR）的变化情况。TPR 代表被正确分类的正样本比例（参见公式 3-8）。FPR 是 FP 相对于实际负样本的比例（参见公式 3-9）。从列联表的元素来看，这些值通过下面的公式计算得到。

$$TPR = \frac{TP}{TP + FN}$$

公式 3-8　真正率

$$FPR = \frac{FP}{TN + FP}$$

公式 3-9　假正率

简单想一下，如果决策阈值使用非常小的值，那么每个样本都会被预测为正例。

此时，TPR=1.0。因为每个样本都被分类为正例，没有假负例（FN=0.0）。FPR=1.0 因为没有例子被分类为负例（TN=0.0）。然而当决策阈值设得很高，TP=0，那么 TPR=0，FP=0，因为没有样本被分类为正例。因此，FPR=0。图 3-10 和图 3-11 使用 pylab roc_curve() 以及 auc() 函数。图 3-10 为基于训练集得到的 ROC 性能曲线。图 3-11 为测试集上的 ROC 性能曲线。

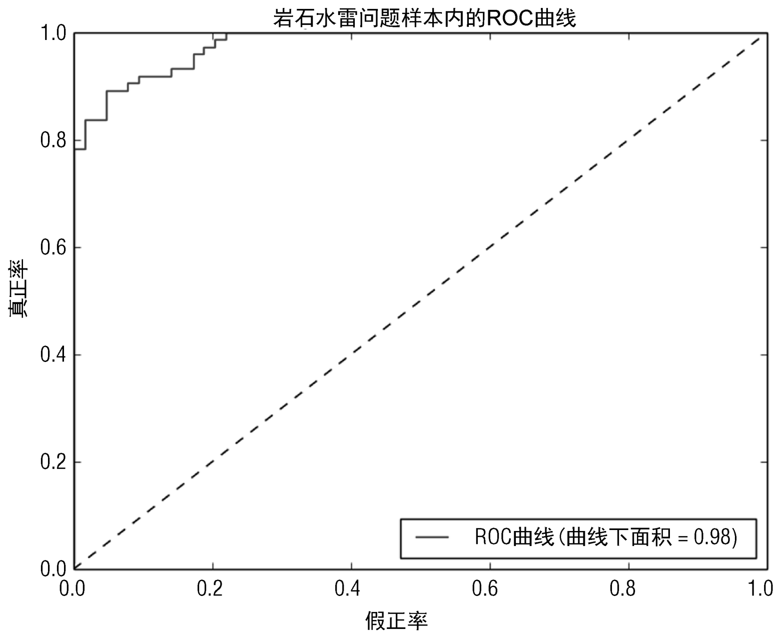


图 3-10　训练集上的岩石 – 水雷分类器的 ROC 曲线

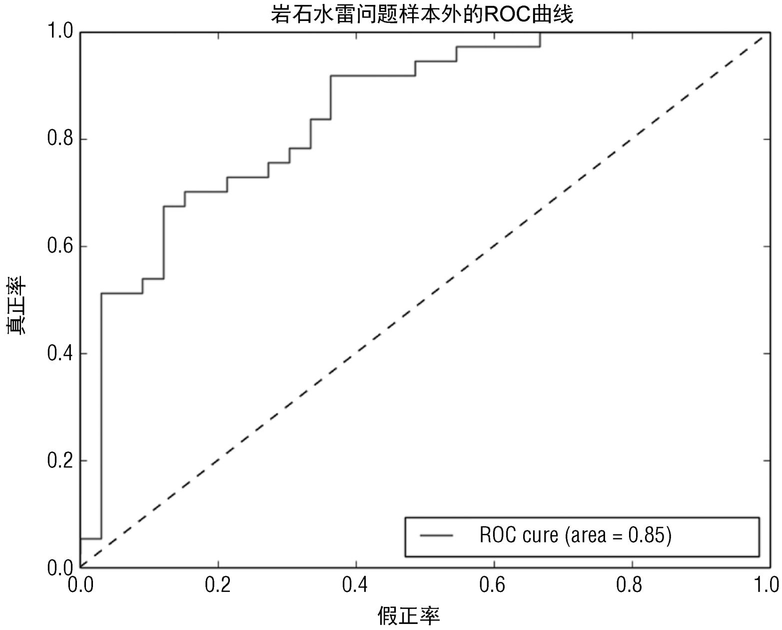


图 3-11 岩石 - 水雷分类器在样本外数据的 ROC 曲线

如果分类器（针对岩石 - 水雷问题）为随机分类，ROC 的样子为一条从左下角到右上角的对角线。这条对角线一般画在图里作为参照点。对于一个完美的分类器，ROC 曲线应该是直接从（0,0）上升到（0,1），然后横着连到（1,1）的直线。显然，图 3-10（在测试集上的结果）要比图 3-11 更接近于完美答案。分类器越接近于左上角，效果越好。如果 ROC 曲线掉到对角线下边，这一般表示数据科学家有可能把预测符号弄反了，此时应该认真检查代码。

图 3-10 和图 3-11 同样展示了曲线下的面积值（AUC）。AUC 正如名字所表达的，指的是 ROC 曲线下的面积。一个完美分类器的 AUC=1.0，随机猜测分类器对应的 AUC 为 0.5。图 3-10 和图 3-11 的 AUC 证明基于训练集进行错误估计往往会高估性能。训练集上的 AUC=0.98，测试集上的 AUC=0.85。

一些用于估算 2 分类问题性能的方法同样适用于多分类问题。误分类错误仍然有意义，混淆矩阵也同样适用。有许多将 ROC 曲线以及 AUC 指标推广到多分类应用的工作。

3.3.2 部署模型的性能模拟

前一节的例子展示了性能评估不能基于训练集进行计算。例子将数据切分为 2 个子集。第一个子集称作训练集，包含 2/3 的可用数据，用于拟合一个普通的最小均方模型。第 2 个子集包含剩下 1/3 的数据，称作测试集，用于评估性能（不在模型训练中使用）。对于

机器学习，以上步骤是一个标准流程。

尽管目前没有明确规则来确定测试集的大小，一般测试集可以占所有数据的 25% ～ 35%。要记住模型训练的性能随着训练集规模的减小而下降。将过多数据从训练集中去掉会影响模型性能估计。

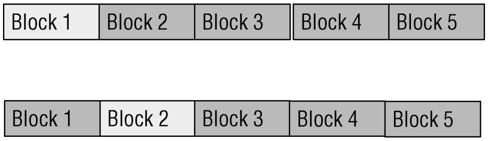


图 3-12　n 折交叉验证

另一种预留数据的方法被称作 n 折交叉验证。图 3-12 展示了如何基于 n 折交叉验证来对数据进行切分。数据集被等分为 n 份不相交的子集。训练和测试需要多次遍历数据。图 3-12 中的 n=5。第一次遍历是，数据的第一块被预留用于测试，剩下 n-1 块用于训练。第 2 遍，第 2 块被预留做测试，剩下 n-1 块用于训练。该过程继续，直到所有数据都被预留一遍（对于图 3-12，5 折交叉验证的样例中，数据会被扫描 5 次）。

n 折交叉验证可以估计预测错误：在多份样本上估计错误来估计错误边界。通过为训练集分配更多样本，生成的模型会产生更低的泛化错误，具备更好的预测性能。例如，如果选择 10 折交叉验证，每次训练只需要留出 10% 的数据进行预测。n 折交叉验证是以更多的训练时间作为代价。保留一个固定的集合作为测试集可以有更快的训练速度，因为它只需要扫描一遍训练数据。当使用 n 折交叉验证的训练时间不可忍受时，使用预留测试集是一个更好的选择，而且如果训练数据很多的话，留一些数据出来不会对模型性能造成太大影响。

另一件值得注意的是测试样本应该能代表整个数据集。上一节例子使用的抽样样本不完全是一个随机样本。它是每隔 3 个样本选 1 个作为测试样本。类似上面方法，通过均匀采样一般足够用了。但必须避免采样过程给训练集和测试集引入偏差。例如，给一类数据，该数据每天生成一个，数据按照采样日期排列，那么 7 折交叉验证使用每隔 7 个点抽样一个点的方法就不正确。

如果研究现象有特殊的统计特征，抽样过程可能要更加小心。此时需要注意在测试样本中保留统计特征。这类例子包括对稀疏事件（如欺诈或者广告点击）进行预测。要建模的事件出现频率非常少，随机抽样可能导致有过多或者过少的样本出现在测试集中，同时导致对性能的错误估计。分层抽样（http://en.wikipedia.org/wiki/Stratified_sampling）将数据切分为不同子集，分别在子集中进行抽样然后组合。如果类别标签对应罕见事件，可能需要分别从欺诈样本以及合法样本中抽样，然后组成测试集。更重要的是，这样的数据就是模型最终要运行的数据。

模型经过训练和测试，那么还应该将训练集和测试集再合并为一个更大的集合，重新在该集合上训练模型。样本外测试已经可以给出预测错误的期望结果。这就是为什么要预

留一部分数据的原因。如果能在更多数据上进行训练，模型效果会更好，泛化能力也更好。真正要部署的模型应该在所有数据上进行训练。本节将提供一些工具来量化模型的预测性能。针对模型以及问题复杂度，下一节将介绍如何使用数值比较来替换图形比较。这种替换使得模型选择过程变得规范。

3.4 模型与数据的均衡

本节使用最小二乘法（OLS）来说明几个问题。首先，它展示为什么 OLS 有时会对问题过拟合。过拟合是指训练数据和测试数据上的错误存在显著差异，比如上一节的 OLS 用于解决岩石 - 水雷分类问题。其次，我们将引入 2 种方法来解决 OLS 的过拟合。这些方法会培养你的直觉，为第 4 章提到的惩罚线性回归方法做铺垫。此外，克服过拟合问题的方法在许多现代机器学习算法中都会用到。现代算法往往会产生大量不同复杂度的模型，然后基于样本外数据的性能来权衡模型复杂度、问题复杂度以及数据集丰富程度，最终决定使用哪个模型。该过程会在后续重复使用。

普通的最小二乘法作为一个原型很好地展示了机器学习算法的方方面面。它是一个有监督的学习算法，包括训练过程以及测试过程。在某些情况下可能会过拟合。最小二乘法与其他现代的函数逼近算法都存在一些共性。然而，与现代机器学习算法相比，OLS 少一个重要特点。在原始公式中（最熟悉的公式），当过拟合发生时，没有办法阻止学习过程。这就像让汽车全速前进（当道路宽敞时很好，在紧急情况下就会有问题）。幸运的是，有大量的工作都在改进最小二乘法，尽管最小二乘法距离当时发明它的高斯和勒让德已经过去了 200 年。本节引入 2 种方法来调整普通最小二乘法的瓶颈：前向逐步回归和岭回归。

3.4.1 通过权衡问题复杂度、模型复杂度以及数据集规模来选择模型

下面一些例子将介绍现代机器学习算法是如何进行调整来更好地拟合问题和数据集。第一个例子是对最小二乘法进行修改，被称作前向逐步回归方法。具体工作过程如下：回忆下公式 3-1 以及公式 3-2 定义要解决的问题（这里对应公式 3-10 以及公式 3-11），向量 Y 包含标签，矩阵 X 包含用于预测标签的属性。

$$Y = \begin{matrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{matrix}$$

公式 3-10 数值标签向量

$$X=\begin{matrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \ddots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{matrix}$$

公式 3-11 数值属性矩阵

如果这是一个回归问题，那么 Y 是包含实数的列向量，线性问题是找到一个权重向量 β 以及一个标量 β_0（参照公式 3-12）。

$$\beta=\begin{matrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{matrix}$$

公式 3-12 线性模型的系数向量

线性模型的目标是选择 β 以更好地逼近 Y（参照公式 3-13）。

$$Y\sim X\beta+\begin{matrix} \beta_0 \\ \beta_0 \\ \vdots \\ \beta_0 \end{matrix}$$

公式 3-13 通过属性的线性函数来近似标签

如果 X 的列数等于 X 的行数，并且 X 不同列之间是相互独立的（不存在相互之间的线性关系），那么 X 可以求逆，符号～可以替换为 =。得到的向量 β 会更精确地拟合标签，看起来很好但不正确。问题在于出现了过拟合（过拟合是指在训练数据上预测效果很好但在新数据上不能复制）。对于真实问题，这并不是一个好的结果。过拟合的根源在于 X 中有太多的列。答案可能是去掉 X 中的一些列。然而去掉一些列又转化为去掉多少列以及哪几列应该去掉的问题。这种蛮力的方法也被称作最佳子集选择。

3.4.2 使用前向逐步回归来控制过拟合

下面代码简要勾勒了最佳子集选择算法的过程。

基本想法是在列的个数上增加一个约束（假设为 nCol），然后从 X 的所有列中抽取特定个数的列构成数据集，在上边执行最小二乘法，遍历所有列的组合（列数为 nCol），找到在测试集上取得最佳效果的 nCol 值；增加 nCol 值，重复上述过程。以上过程产生最佳的一列子集、两列子集一直到所有列子集（对应矩阵 X）。对于每个子集同样有一个性能与

之对应。下一步决定在部署时是使用一列子集版本、两列子集版本，还是其他版本。到这就简单了，直接选择错误率最低的版本。

```
Initialize: Out_of_sample_error = NULL
   Break X and Y into test and training sets
for i in range(number of columns in X):
   for each subset of X having i+1 columns:
      fit ordinary least squares model
   Out_of_sample_error.append(least error amoung subsets containing
   i+1 columns)
Pick the subset corresponding to least overall error
```

最佳子集选择存在的一个问题是该算法需要大量计算，即使属性不多的情况下（属性数对应 *X* 的列数），计算量也非常巨大。例如，10 个属性对应于 2^{10}=1000 个子集。有几种方法可以避免这种情况。下面的代码展示了前向逐步回归的过程。前向逐步回归的想法是从 1 列子集开始，找到效果最佳的那一列属性，接着寻找与其组合与效果最佳的第 2 列属性，而不是评估所有的 2 列子集。前向逐步回归的伪代码如下。

```
Initialize: ColumnList = NULL
   Out-of-sample-error = NULL
   Break X and Y into test and training sets
For number of column in X:
   For each trialColumn (column not in ColumnList):
      Build submatrix of X using ColumnList + trialColumn
      Train OLS on submatrix and store RSS Error on test data
   ColumnList.append(trialColumn that minimizes RSS Error)
   Out-of-sample-error.append(minimum RSS Error)
```

最佳子集选择以及前向逐步回归过程基本类似。它们训练一系列的模型（列数为 1 训练几个，列数为 2 训练几个，等等）。这种方法产生了参数化的模型族（所有线性回归以列数作为参数）。这些模型在复杂度上存在差异，最后的模型通过在预留样本上计算错误进行选择。

代码清单 3-3 为在红酒数据集上实现的前向逐步回归的 Python 代码。

代码清单 3-3 前向逐步回归：红酒品质数据 -fwdStepwiseWine.py

```
import numpy
from sklearn import datasets, linear_model
from math import sqrt
import matplotlib.pyplot as plt
```

```
def xattrSelect(x, idxSet):
    #takes X matrix as list of list and returns subset containing
    #columns in idxSet
    xOut = []
    for row in x:
        xOut.append([row[i] for i in idxSet])
    return(xOut)

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learningdatabases/"
"wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)
xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

#divide attributes and labels into training and test sets
indices = range(len(xList))
xListTest = [xList[i] for i in indices if i%3 == 0 ]
xListTrain = [xList[i] for i in indices if i%3 != 0 ]
labelsTest = [labels[i] for i in indices if i%3 == 0]
labelsTrain = [labels[i] for i in indices if i%3 != 0]
```

```
#build list of attributes one-at-a-time - starting with empty
attributeList = []
index = range(len(xList[1]))
indexSet = set(index)
indexSeq = []
oosError = []

for i in index:
    attSet = set(attributeList)
    #attributes not in list already
    attTrySet = indexSet - attSet
    #form into list
    attTry = [ii for ii in attTrySet]
    errorList = []
    attTemp = []
    #try each attribute not in set to see which
    #one gives least oos error
    for iTry in attTry:
        attTemp = [] + attributeList
        attTemp.append(iTry)
        #use attTemp to form training and testing sub matrices
        #as list of lists
        xTrainTemp = xattrSelect(xListTrain, attTemp)
        xTestTemp = xattrSelect(xListTest, attTemp)
        #form into numpy arrays
        xTrain = numpy.array(xTrainTemp)
        yTrain = numpy.array(labelsTrain)
        xTest = numpy.array(xTestTemp)
        yTest = numpy.array(labelsTest)
        #use sci-kit learn linear regression
        wineQModel = linear_model.LinearRegression()
        wineQModel.fit(xTrain,yTrain)
        #use trained model to generate prediction and calculate rmsError
        rmsError = numpy.linalg.norm((yTest-wineQModel.predict(xTest)),
            2)/sqrt(len(yTest))
        errorList.append(rmsError)
        attTemp = []

    iBest = numpy.argmin(errorList)
    attributeList.append(attTry[iBest])
```

```
    oosError.append(errorList[iBest])

print("Out of sample error versus attribute set size" )
print(oosError)
print("\n" + "Best attribute indices")
print(attributeList)
namesList = [names[i] for i in attributeList]
print("\n" + "Best attribute names")
print(namesList)

#Plot error versus number of attributes
x = range(len(oosError))
plt.plot(x, oosError, 'k')
plt.xlabel('Number of Attributes')
plt.ylabel('Error (RMS)')
plt.show()

#Plot histogram of out of sample errors for best number of attributes
#Identify index corresponding to min value,
#retrain with the corresponding attributes
#Use resulting model to predict against out of sample data.
#Plot errors (aka residuals)
indexBest = oosError.index(min(oosError))
attributesBest = attributeList[1:(indexBest+1)]

#Define column-wise subsets of xListTrain and xListTest
#and convert to numpy
xTrainTemp = xattrSelect(xListTrain, attributesBest)
xTestTemp = xattrSelect(xListTest, attributesBest)
xTrain = numpy.array(xTrainTemp); xTest = numpy.array(xTestTemp)

#train and plot error histogram
wineQModel = linear_model.LinearRegression()
wineQModel.fit(xTrain,yTrain)
errorVector = yTest-wineQModel.predict(xTest)
plt.hist(errorVector)
plt.xlabel("Bin Boundaries")
plt.ylabel("Counts")
plt.show()
```

```
#scatter plot of actual versus predicted
plt.scatter(wineQModel.predict(xTest), yTest, s=100, alpha=0.10)
plt.xlabel('Predicted Taste Score')
plt.ylabel('Actual Taste Score')
plt.show()
```

上面的代码清单包含一个函数用于从 *X* 矩阵中抽取选择的列（对应于 Python 的列表 list，该列表的每个元素也是一个列表）。然后该函数将 *X* 矩阵与标签向量划分为训练集和测试集。之后，代码完成前面描述的算法。算法的遍历从属性的一个子集开始。第一遍时，该子集为空。对于后续的遍历，该子集包含上一次遍历选择的属性。每一次遍历都会选择一个新的属性添加到属性子集中。待添加的属性是通过对每一个非包含的属性进行测试：选择添加属性以后性能提高最多的属性。每一个属性被加入属性子集以后，使用普通的最小二乘法来拟合模型。对每一个测试属性，在预留样本上评估性能。产生最佳根损失（RSS）的属性被加入属性集，关联的 RSS 错误也会进行计算。

图 3-13 为 RMSE 与用于回归的属性个数之间的函数关系。在 9 个属性全部包含进来以前，错误一直在降低，然后增加。

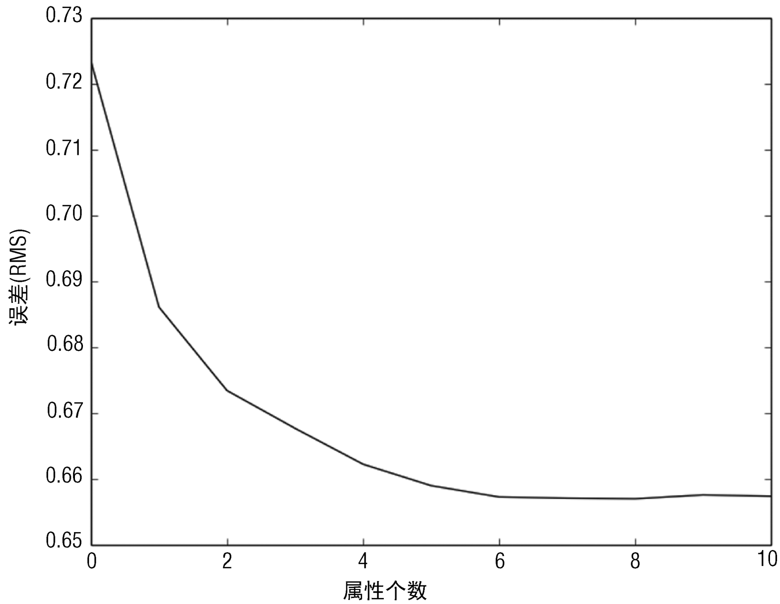


图 3-13 使用前向逐步回归方法获得的红酒品质预测

代码清单 3-4 为前向逐步回归方法应用于酒品质量预测的数值输出。

代码清单 3-4 前向逐步回归的输出 -fwdStepwiseWineOutput.txt

```
Out of sample error versus attribute set size
[0.7234259255116281, 0.68609931528371915, 0.67343650334202809,
0.66770332138977984, 0.66225585685222743, 0.65900047541546247,
0.65727172061430772, 0.65709058062076986, 0.65699930964461406,
0.65758189400434675, 0.65739098690113373]

Best attribute indices
[10, 1, 9, 4, 6, 8, 5, 3, 2, 7, 0]

Best attribute names
['"alcohol"', '"volatile acidity"', '"sulphates"', '"chlorides"',
 '"total sulfur dioxide"', '"pH"', '"free sulfur dioxide"',
 '"residual sugar"', '"citric acid"', '"density"', '"fixed acidity"']
```

第一个列表（python 的 list 对象）展示了 RSS 错误。错误一直降低，直到将第 10 个元素加入列表，然后错误变高。关联的列索引在后一个列表中给出。最后的列表给出了关联属性的名称（列名）。

3.4.3 评估并理解你的预测模型

其他几个图对于理解一个学习好的算法性能非常有帮助，这些图指出了性能提升的途径。图 3-14 为测试集上每个点的实际标签值与预测标签值的散点图。在理想情况下，图 3-14 中的所有点会分布在 45 度线上，这条线上的真正标签与预测标签是相等的。因为真正得分是整数，所以散点图分布在水平方向上。如果真正标签分布在少量的数值上，将每个数据点绘制成半透明状会很有用，一个区域的颜色深度就能反映点的堆积程度。对得分在 5 和 6 上的实际酒品的预测结果非常好。对更极端的值，系统预测效果也不好。一般来讲，机器学习算法对边缘数据的预测效果并不好。

图 3-15 为前向逐步预测算法对酒品预测的错误直方图。有时错误直方图会有 2 个甚至多个离散的波峰，比如在最右边或者最左边有一个小的波峰。在这种情况下，可以继续寻找错误中不同波峰的解释，添加能够辨识归类的新属性来降低预测错误。

对于上面的输出结果要记住以下几点。首先，重新回顾一下整个学习过程，这里过程是指训练一组模型（在本例中，模型对应于基于 X 的列子集的普通线性回归）。这一系列模型进行了参数化（本例中，通过线性模型中的属性个数进行区分）。最终选择的模型在样本外的错误最小。解决方案中引入的属性个数称作复杂度参数。复杂度更高的模型会有更多自由参数，相对于低复杂度的模型更容易对数据产生过拟合。

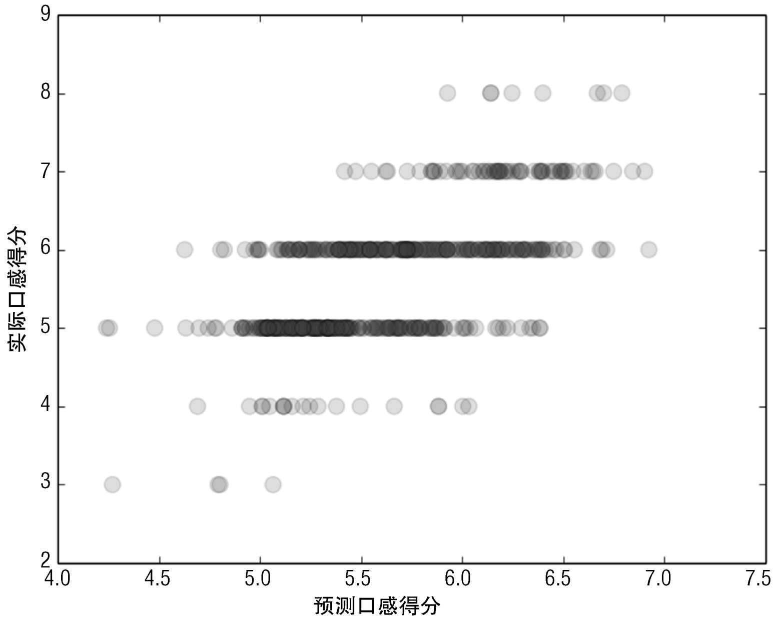


图 3-14 实际口感得分与前向逐步回归生成的预测口感得分的比较

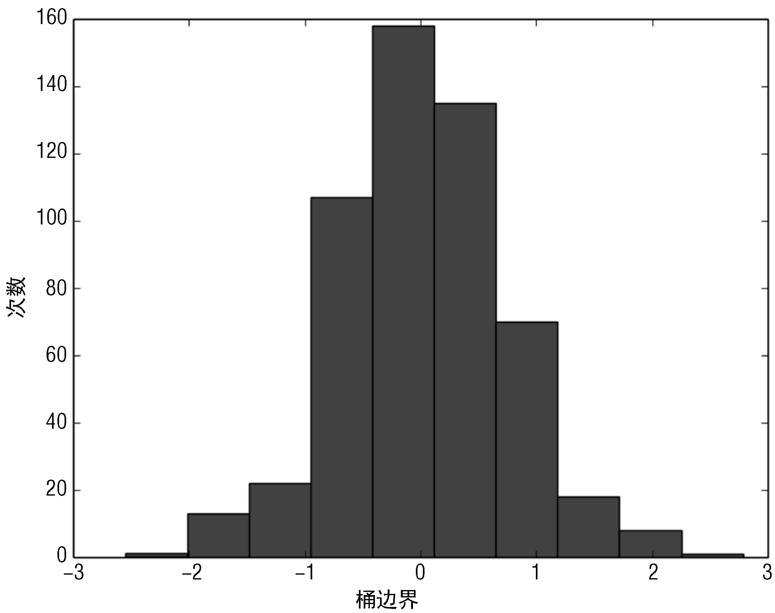


图 3-15 使用前向逐步回归得到的红酒口感预测直方图

另外注意到属性已经根据其对预测的重要性进行了排序。在包含列编号的 List 以及

属性名的 List 中，第一个元素是第一个选择的属性，第二个元素是第二个选择的属性，以此类推。用到的属性按顺序排列。这是机器学习任务中一个很重要并且必需的特征。早期机器学习任务大部分都包括寻找（或者构建）用于构建预测的最佳属性集。而能够对属性进行排序的算法对于上述任务非常有帮助。本书中介绍的其他算法也具备本特点。

最后一点是挑选模型。模型越复杂，泛化能力越差。在同等情况下，倾向于选择不太复杂的模型。前面例子表明从第 9 个模型到第 10 个模型的性能下降幅度很小（只在第 4 位有变化）。最佳经验是如果属性添加后带来的性能提升只达到小数点后第 4 位，那么保守起见，可以将这样的属性移除掉。

3.4.4 通过惩罚回归系数来控制过拟合——岭回归

本节描述了另外一种通过修改最小二乘法来控制模型复杂度从而避免过拟合的方法。这也是第一次介绍惩罚线性回归方法。第 4 章会更详细地介绍。

普通最小二乘法的目标是找到能够满足公式 3-14 的标量 β_0 以及向量 β。

$$\beta_0^*, \beta^* = \arg\min_{\beta_0,\beta}\left(\frac{1}{m}\sum_{i=1}^{m}(y_i - (\beta_0 + x_i\beta))^2\right)$$

公式 3-14 OLS 对应的最小化问题

符号 argmin 是指“能够最小化表达式的 β_0 以及 β”。系数 β_0* 以及 β* 是最小二乘法的解。最佳子集回归以及前向逐步回归通过限制使用的属性个数来控制回归的复杂度。另外一种方法称作惩罚系数回归。惩罚系数回归是使系数变小，而不是将其中一些系数设为 0。一种惩罚线性回归的方法被称作岭回归。岭回归问题的定义见公式 3-15。

$$\beta_0^*, \beta^* = \arg\min_{\beta_0,\beta}\left(\frac{1}{m}\sum_{i=1}^{m}(y_i - (\beta_0 + x_i\beta))^2 + \alpha\beta^T\beta\right)$$

公式 3-15 岭回归对应的最小化问题

公式 3-15 以及普通最小二乘法（见公式 3-14）的差别在 $\alpha\beta^T\beta$ 项上。$\beta^T\beta$ 项是 β（系数向量）的欧几里得范数的平方。变量 β 是这类问题的复杂度参数。如果 $\alpha = 0$，问题变为普通最小二乘法。如果 α 变大，β（系数的瓶阀）接近于 0，那么通过常数项 β_0 就可以预测标签 y_i。scikit-learn 包给出了岭回归的实现。代码清单 3-5 为使用岭回归来解决红酒口感回归问题的代码。

代码清单 3-5 使用岭回归预测红酒口感 -ridgeWine.py

```
__author__ = 'mike-bowles'
```

```
import urllib2
import numpy
from sklearn import datasets, linear_model
from math import sqrt
import matplotlib.pyplot as plt

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learningdatabases/"
"wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

#divide attributes and labels into training and test sets
indices = range(len(xList))
xListTest = [xList[i] for i in indices if i%3 == 0 ]
xListTrain = [xList[i] for i in indices if i%3 != 0 ]
labelsTest = [labels[i] for i in indices if i%3 == 0]
labelsTrain = [labels[i] for i in indices if i%3 != 0]

xTrain = numpy.array(xListTrain); yTrain = numpy.array(labelsTrain)
xTest = numpy.array(xListTest); yTest = numpy.array(labelsTest)
```

```
alphaList = [0.1**i for i in [0,1, 2, 3, 4, 5, 6]]

rmsError = []
for alph in alphaList:
    wineRidgeModel = linear_model.Ridge(alpha=alph)
    wineRidgeModel.fit(xTrain, yTrain)
    rmsError.append(numpy.linalg.norm((yTest-wineRidgeModel.predict(
        xTest)), 2)/sqrt(len(yTest)))

print("RMS Error              alpha")
for i in range(len(rmsError)):
    print(rmsError[i], alphaList[i])

#plot curve of out-of-sample error versus alpha
x = range(len(rmsError))
plt.plot(x, rmsError, 'k')
plt.xlabel('-log(alpha)')
plt.ylabel('Error (RMS)')
plt.show()

#Plot histogram of out of sample errors for best alpha value and
#scatter plot of actual versus predicted

#Identify index corresponding to min value, retrain with
#the corresponding value of alpha

#Use resulting model to predict against out of sample data.
#Plot errors (aka residuals)
indexBest = rmsError.index(min(rmsError))
alph = alphaList[indexBest]
wineRidgeModel = linear_model.Ridge(alpha=alph)
wineRidgeModel.fit(xTrain, yTrain)
errorVector = yTest-wineRidgeModel.predict(xTest)
plt.hist(errorVector)
plt.xlabel("Bin Boundaries")
plt.ylabel("Counts")
plt.show()

plt.scatter(wineRidgeModel.predict(xTest), yTest, s=100, alpha=0.10)
plt.xlabel('Predicted Taste Score')
```

```
plt.ylabel('Actual Taste Score')
plt.show()
```

回忆一下前向逐步回归算法生成了一系列不同的模型，第一个模型只包含一个属性，第二个模型包含两个属性，等等，直到最后的模型包含所有属性。岭回归代码也包含一系列的模型。岭回归通过不同的 α 值来控制模型数量，而不是通过属性个数来控制。α 参数决定了对 β 的惩罚力度。α 的一系列值按照 10 的倍数递减。一般来讲，你希望 α 按照指数级进行递减，并不是按照一个固定的增量。α 的取值范围一般要设置得足够宽，往往需要通过实验来确定。

图 3-16 为 RMSE 与岭回归的复杂度参数 α 的对应关系。参数按照从左到右、从大到小排列。传统习惯是一般在左边绘制简单模型，在右边绘制复杂模型。图 3-16 显示了与逐步前向回归类似的特点。错误几乎是一样的，但前向逐步回归的错误相比来说更大一些。

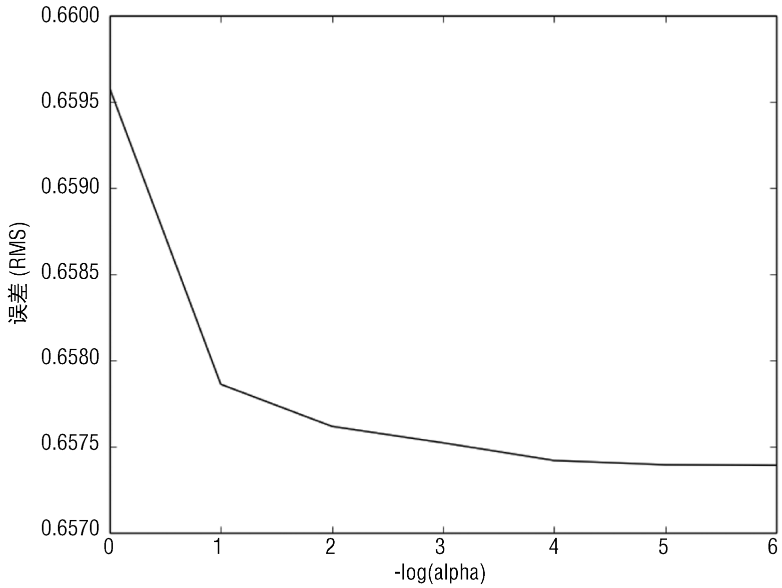


图 3-16 使用岭回归预测红酒品质

代码清单 3-6 展示了来自岭回归的输出。数字显示岭回归与前向逐步回归有几乎相同的特点。数据更加支持前向逐步回归。

代码清单 3-6 岭回归输出 -ridgeWineOutput.txt

```
RMS Error             alpha
(0.65957881763424564, 1.0)
```

```
(0.6578610918808592 8, 0.1)
(0.6576172144640245 5, 0.010000000000000002)
(0.6575216482641753 6, 0.0010000000000000002)
(0.6574190680109293 1, 0.00010000000000000002)
(0.6573941628851253 1, 1.0000000000000003e-05)
(0.6573913087155859 3, 1.0000000000000004e-06)
```

图 3-17 为在红酒数据集上使用岭回归的实际口感得分与预测得分的散点图。图 3-18 为预测错误的直方图。

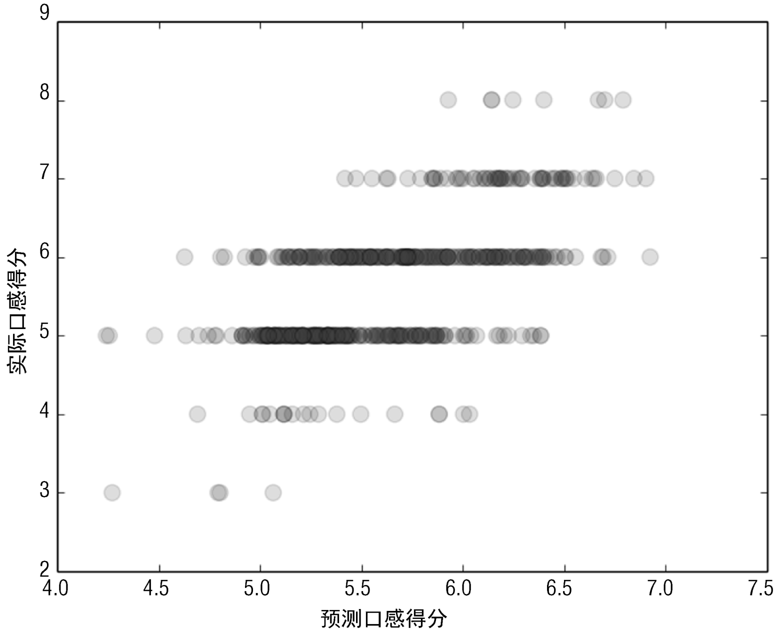


图 3-17 使用岭回归的实际口感得分与预测得分的散点图

也可以使用更通用的方法来解决分类问题。3.3“度量预测模型性能”讨论了量化分类性能的方法，包括使用误分类错误、不同预测结果的经济代价以及曲线（ROC 曲线）下面积 AUC 来量化性能。

本节使用普通最小二乘法来构建分类器。代码清单 3-7 为相同类型的 Python 代码。与 OLS 不同，它使用岭回归作为回归方法（使用一个复杂度控制参数）来构建岩石 - 水雷分类器，使用 AUC 作为分类器的性能度量。代码清单 3-7 类似于红酒品质预测的代码。一个大的区别是代码清单 3-7 使用在测试数据上的预测以及对应的实际结果作为函数 roc_curve（scikit-learn 包中的函数）的输入。这会使得每次训练完成后，AUC 的计算变得更

加简单。每个点的错误值经过累加，生成的性能指标被打印出来（参见代码清单 3-8）。

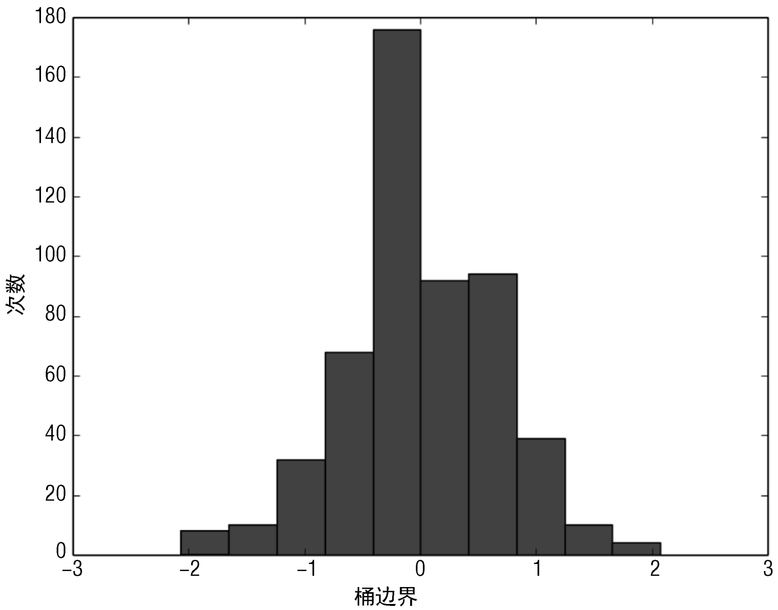


图 3-18 使用岭回归得到的口感预测的错误直方图

代码清单 3-7 使用岭回归进行岩石 - 水雷分类 -classifierRidgeRocksVMines.py

```
__author__ = 'mike-bowles'
import urllib2
import numpy
from sklearn import datasets, linear_model
from sklearn.metrics import roc_curve, auc
import pylab as plt

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-earning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []
labels = []
for line in data:
```

```
    #split on comma
    row = line.strip().split(",")
    #assign label 1.0 for "M" and 0.0 for "R"
    if(row[-1] == 'M'):
        labels.append(1.0)
    else:
        labels.append(0.0)
        #remove lable from row
    row.pop()
    #convert row to floats
    floatRow = [float(num) for num in row]
    xList.append(floatRow)

#divide attribute matrix and label vector into training(2/3 of data)
#and test sets (1/3 of data)
indices = range(len(xList))
xListTest = [xList[i] for i in indices if i%3 == 0 ]
xListTrain = [xList[i] for i in indices if i%3 != 0 ]
labelsTest = [labels[i] for i in indices if i%3 == 0]
labelsTrain = [labels[i] for i in indices if i%3 != 0]

#form list of list input into numpy arrays to match input class for
#scikit-learn linear model
xTrain = numpy.array(xListTrain); yTrain = numpy.array(labelsTrain)
xTest = numpy.array(xListTest); yTest = numpy.array(labelsTest)

alphaList = [0.1**i for i in [-3, -2, -1, 0,1, 2, 3, 4, 5]]

aucList = []
for alph in alphaList:
    rocksVMinesRidgeModel = linear_model.Ridge(alpha=alph)
    rocksVMinesRidgeModel.fit(xTrain, yTrain)
    fpr, tpr, thresholds = roc_curve(yTest,rocksVMinesRidgeModel.
        predict(xTest))
    roc_auc = auc(fpr, tpr)
    aucList.append(roc_auc)


print("AUC alpha")
for i in range(len(aucList)):
```

```
    print(aucList[i], alphaList[i])

#plot auc values versus alpha values
x = [-3, -2, -1, 0,1, 2, 3, 4, 5]
plt.plot(x, aucList)
plt.xlabel('-log(alpha)')
plt.ylabel('AUC')
plt.show()

#visualize the performance of the best classifier
indexBest = aucList.index(max(aucList))
alph = alphaList[indexBest]
rocksVMinesRidgeModel = linear_model.Ridge(alpha=alph)
rocksVMinesRidgeModel.fit(xTrain, yTrain)

#scatter plot of actual vs predicted
plt.scatter(rocksVMinesRidgeModel.predict(xTest),
    yTest, s=100, alpha=0.25)
plt.xlabel("Predicted Value")
plt.ylabel("Actual Value")
plt.show()
```

代码清单 3-8 为 AUC 以及对应的 alpha 值（系数惩罚项的因子）。

代码清单 3-8　使用岭回归得到的岩石 - 水雷分类模型的输出

```
AUC                alpha
(0.84111384111384113, 999.9999999999999)
(0.86404586404586403, 99.999999999999999)
(0.9074529074529073, 10.0)
(0.91809991809991809, 1.0)
(0.88288288288288286, 0.1)
(0.8615888615888615, 0.010000000000000002)
(0.85176085176085159, 0.0010000000000000002)
(0.85094185094185093, 0.00010000000000000002)
(0.84930384930384917, 1.0000000000000003e-05)
```

AUC 的值接近 1 对应于更好的性能，接近于 0.5 说明效果不太好。使用 AUC 的目标是使其最大化而不是最小化，可以参照之前例子中的 MSE。AUC 在 α =1.0 时有一个明显

的突起。数据以及图示显示在 α 远离 1.0 时有明显的下降。回忆一下随着 α 变小，解方案接近于不受限的线性回归问题。α 小于 1.0 时，性能下降表明不受限的解难以达到岭回归的效果。在 3.3 节中，可以看到不受限普通均方回归的结果，其中 AUC 在训练集上的预测性能为 0.98，在测试集上的预测性能为 0.85，非常接近于使用较小的 alpha 的岭回归（α 设为 1E-5）的 AUC 值。这说明岭回归会显著提升性能。

对于岩石 - 水雷问题，数据集包含 60 个属性，总共 208 行数据。将 70 个样本移除作为预留数据，剩下 138 行用于训练。样本数量大约是属性数量的 2 倍，但是不受限解（基于普通的最小二乘法）仍然会过拟合数据。这时使用 10 折交叉验证来估计性能是一个很好的替换方案。使用 10 折交叉验证，每一份数据只有 20 个样本，训练数据相对测试数据就会多很多，从而性能上会有一致的提升。该方法将在第 5 章讨论。

图 3-19 为 AUC 与 alpha 参数的关系，该图展示了在系数向量上使用欧式长度限制可以降低解的复杂度。

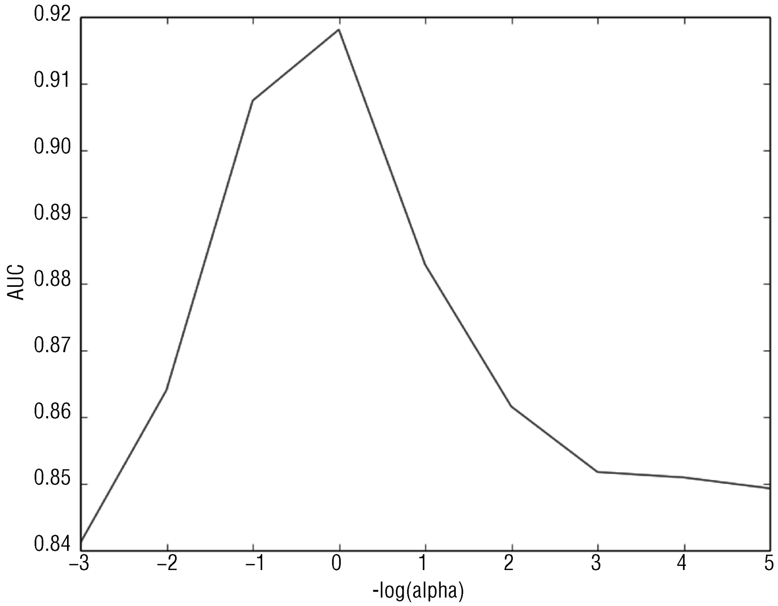


图 3-19 岭回归分类器在岩石 - 水雷数据上得到的 AUC 曲线

图 3-20 为实际分类结果与分类器预测结果的散点图。该图与红酒预测中的散点图类似。因为实际预测的输出是离散的，所以呈现 2 行水平的点。

本节介绍并探索了普通最小二乘法的 2 种扩展方法、训练以及选择一个现代预测模型的过程。这些扩展方法的介绍有助于理解更普通的惩罚线性回归方法（将在第 4 章中介绍），

第 5 章会应用这些方法来解决多个问题。

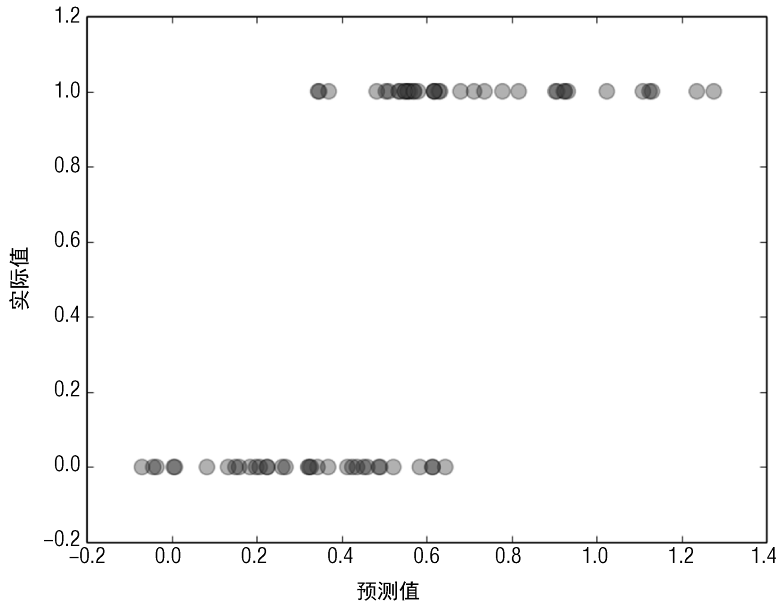


图 3-20　对岩石 − 水雷问题使用岭回归分类器得到的实际结果与预测结果的关系图

小结

本章首先给出了围绕问题复杂性以及模型复杂性的可视化示例，讨论了这些因素以及数据集大小如何影响给定问题的分类性能。接着讨论了针对不同问题（回归、分类以及多分类）度量预测性能的多个评价指标，这些指标也是函数逼近问题的一部分。介绍了用于在新数据上评估性能的 2 种方法（在测试集上评估以及 n 折交叉验证）、机器学习生成一族参数化模型的框架，以及如何基于测试集的性能来选择实际应用的模型。后续以普通最小二乘法为例，介绍相关的概念框架。

参考文献

David J. Hand and Robert J. Till（2001）. A Simple Generalization of the Area Under the ROC Curve for Multiple Class Classification Problems . Machine Learning , 45(2), 171 – 186 .

第 4 章

惩罚线性回归模型

正如第 3 章中所看到的，线性回归在实际应用时需要对普通最小二乘法进行一些修改。普通最小二乘法只在训练数据上最小化错误，难以顾及所有数据。第 3 章的例子展示了普通最小二乘法在新数据集上的效果要远远差于在训练集上的效果，以及普通最小二乘法的 2 种扩展方法。这 2 种方法都涉及减少用于训练最小二乘法的数据集规模以及预留一部分样本来对性能进行评估，从而挑选最佳模型。

前面逐步回归基于普通最小二乘法，首先只使用一列属性来构建预测，挑选效果最好的一列。后续继续添加新的属性到现有模型中。

岭回归引入了一种完全不同的限制。岭回归对参数维度进行惩罚从而对解进行限制。岭回归以及前向逐步回归在样例问题上的效果要明显好于普通最小二乘法。

本章将介绍一族用于克服最小二乘法（OLS）过拟合问题的方法。这些方法称作惩罚线性回归方法。第 3 章介绍的岭回归是惩罚线性回归的一个特例。岭回归通过对回归系数的平方和进行惩罚来避免过拟合。其他惩罚回归算法使用不同形式的惩罚项。本章将介绍惩罚方法是如何确定解的范围以及解的类型。

4.1 为什么惩罚线性回归方法如此有效

下面几个特点使得惩罚线性回归方法非常有效。

- 模型训练足够快速。
- 变量的重要性信息。
- 部署时的预测足够快速。
- 在各种问题上性能可靠，尤其对样本并不明显多于属性的属性矩阵，或者非常稀疏的矩阵。希望模型为稀疏解（即只使用部分属性进行预测的吝啬模型）。
- 问题可能适合使用线性模型来解决。

这就是作为机器学习模型设计者应该了解的关于线性模型的特点。

4.1.1 足够快速地估计系数

训练时间的重要性体现在下面几个方面。一方面是因为模型的构建往往是迭代进行的。你会发现模型训练是特征选择以及特征工程的基础。你会挑选一些看起来合理的特征来训练模型并且在预留数据上评估模型，接下来想继续提升性能，你会做些修改，然后重复尝试上面过程。如果基本的训练可以很快完成，那么就不会浪费太多时间来等结果（如果喝咖啡的话，你就会少摄入太多的咖啡因，提升健康）。这会使得开发过程加快。另一方面是如果条件改变的话，可能需要重新训练模型。如果你在分类微博消息，模型可能需要与词汇的更新同步。如果你在训练面向金融市场的自动交易模型，条件会一直在变。即使不考虑特征重构，训练时间的多少也会决定你的应变速度。

4.1.2 变量的重要性信息

本书涵盖的算法类型可以导出变量的重要性信息。变量的重要性信息包括对模型属性进行排序。属性顺序表明其对模型的价值。排序高的属性要比排序低的属性对模型准确度的贡献更大。变量重要性是一个关键信息。首先，该信息在特征工程中有助于对属性进行剪枝。好的特征会排到列表前面，应该保留，不太好的特征会排到最后，构建模型时可以去掉除了对特征工程有帮助，了解哪些变量在驱动着预测结果可以帮助你去更好地理解模型以及向其他人（你的老板、你的客户以及公司领域专家等）解释模型。属性重要性与人们的期望越靠近，人们对模型的效果越有信心。如果一些排序比较奇怪，你可能对问题会有新的认识。讨论关于属性的重要性可以为提升你的开发团队的性能带来新的启发。

借由快速训练以及计算变量重要性方面的优势，对于任何新的问题都可以先尝试惩罚线性回归方法，这可以使你快速了解问题，并决定哪些特征是有用的。

4.1.3 部署时的预测足够快速

对一些问题来讲，快速计算预测结果是一个关键的性能参数。在一些电子市场（如互联网广告以及自动交易），先得到答案就会先获利。对于许多其他应用（如垃圾过滤），尽管答案并非严格是否，预测时间的快慢也很重要。不论哪种算法，其预测速度很难超越线性模型。线性模型在预测时，仅需要对包含的每个属性进行一次相乘以及一次相加操作。

4.1.4 性能可靠

性能可靠意味着惩罚线性回归方法对不同数据分布及不同数据规模的问题都会产生一个较好的解。对于一些问题，性能最佳。在部分情况下，使用一点技巧，方法就会超过其他所有模型。本章会对这些技巧做一些讨论。第 6 章也会讨论该话题，并介绍使用惩罚线性回归以及集成方法来提升性能的思路。

4.1.5 稀疏解

稀疏解意味着模型中的许多系数等于 0，这也意味着在线预测时，相乘以及相加的次数会减少。更重要的是，稀疏模型（非 0 的系数较少）更容易解释，即更容易看到模型中的哪些属性在驱动着预测结果。

4.1.6 问题本身可能需要线性模型

最后一个使用惩罚线性回归的原因是线性模型可能是解决方案本身的需要。保险支付可以作为需要线性模型的一个例子，其中合同往往包含支付公式，而公式本身又包含变量以及系数。如果使用集成模型，其中每棵树有一千个参数、整体包含数千棵树，那么这样的模型几乎不可能用文字解释清楚。医药测试是另一个需要使用线性形式进行统计推断的例子。

4.1.7 什么时候使用集成方法

不使用惩罚线性回归的主要原因是使用其他技术可能获得更好的性能，比如集成方法。正如第 3 章指出的，集成方法对复杂问题（如极度不规则的决策曲面）或者可以利用大量数据进行求解的问题性能表现最佳。此外，集成方法在度量变量重要性时，可以生成更多关于属性与预测结果关系的信息。例如，集成方法会发现 2 阶甚至更高阶的重要性信息，即哪些变量组合的重要性大于单独对这些变量的重要性加和。这些信息可以在惩罚线性回归的基础上进一步提升性能。第 6 章会详细介绍这一点。

4.2 惩罚线性回归：对线性回归进行正则化以获得最优性能

正如第 3 章所讨论的，本书解决了一类称作函数逼近的问题。训练模型的起始点是包含大量样本的数据集。每个实例包含结果以及大量用于预测结果的属性。第 3 章给出了一个简单的例子。表 4-1 为一个稍微修改的例子。

表 4-1 样例训练集

结　　果	特征 1	特征 2	特征 3
2013 年的花费	性别	2012 年的划分	年龄
100	M	0	25
225	F	260	32
75	F	12	17

因为表 4-1 中的输出是实数值，所以该问题是一个回归问题。性别属性（特征 1）只能取 2 个值，所以该属性为类别属性（或者方面）属性。其他 2 个属性是数值属性。函数逼近的目标是：构建一个从属性到输出的函数；在某种意义下最小化错误。第 3 章讨

论过一些其他的错误计算公式，这些备选方案都可以用于量化整体错误。

表 4-1 所示的数据集经常被表示为一个包含结果的列向量以及一个包含属性的矩阵（3 列特征）。将结果列向量与矩阵合并在数学上容易混淆。严格来讲，矩阵包含的元素要定义在相同的数据类型上。矩阵内容可以是实数、整数、复数以及二元数字等，但它们不能是实数与类别变量的混合。

这里有一个要点需要记住：线性方法只能操作数值属性。表 4-1 中的数据包含非数值数据，因此线性方法不能直接应用。幸运的是，将表 4-1 中的数据转换为数值数据相对简单。在 4.4.4 节中，我们会学习将类别属性转换为数值属性的编码技术。如果属性全部为实数值（不论是最初问题定义，还是通过将类别属性转换为实数值），那么线性回归问题的数据可以通过 2 个对象来表示：Y 和 X，其中 Y 是一个包含结果的列向量，X 是一个实数矩阵。

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

公式 4-1 结果向量

对于表 4-1 中的例子，Y 是结果列。

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & & \ddots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix}$$

公式 4-2 属性矩阵

对于表 4-1 的例子，X 是将结果列排除后的剩余列集合。

Y（y_i）的第 i 个元素对应于 X 的第 i 行。X 的第 i 行使用包含下标的 x_i 表示，x_i=（x_i1,x_i2,x_im）。普通最小二乘法的目标是最小化 y_i 与 x_i（X 的第 i 行属性）的线性函数之间的错误，即找到实数向量 β 以及标量 β_0，从而使得来自 Y 的每个元素 y_i 可以通过公式 4-4 来近似。

$$\beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{pmatrix}$$

公式 4-3 β– 模型系数向量

$$\begin{aligned}\text{Prediction of } y_i &= x_i * \beta + \beta_0 \\ &= x_{i1} * \beta_1 + x_{i2} * \beta_2 + \cdots + x_{im} * \beta_m + \beta_s\end{aligned}$$

公式 4-4 *X* 与 *Y* 的线性关系

你也可以利用你的经验知识来寻找到 β 值。在表 4-1 中，可能你会觉得人们 2013 年会比 2012 年多消费 10 美元，即他们购买量年均增长 10%，甚至新生儿也会购买 50 美元的书。这些信息就可以构建图书消费的预测公式，类似公式 4-5。

2013 年的花销 =\$50 + 1.1*（\$ 2012 年的花销）+ \$10 *Age

公式 4-5 预测图书销售

公式 4-5 并不使用性别属性，因为它是一个类别变量（这方面处理会在“引入非数值属性到线性方法”中介绍，目前暂时忽略）。公式 4-5 生成的预测并不能精确匹配表 4-1 中的结果（实际数量）。

4.2.1 训练线性模型：最小化错误以及更多

即使你能手动检查 β 的值，一般手动寻找 β 值也不是最佳方式。对于许多问题，变量数目以及变量间的关系使得猜测 β 的值不可行。所以一般方法是通过解最小化问题来找到属性乘子。最小化问题是找到使得均方误差最小（但不是 0）的 β 值。

公式 4-4 的两边完全相等就意味着模型已经过拟合。公式 4-4 的右侧是你要训练的预测模型。该模型的含义是：为了构建一个预测，把每一个属性乘以对应的系数，加起来再加一个常数。训练意味着找到构成向量 β 以及常量 β_0 的数值。错误被定义为 y_i 的实际值与 y_i 的预测值之间的差异（见公式 4-4）。均方误差是将所有样本的错误归约到一个数。之所以选择错误的平方是因为错误不区分正负，而平方函数从数学上求解更加方便。普通最小二乘法的定义变为找到 β* 以及 β_0*（上标 * 表明这些值是 β 的最佳值），能够满足公式 4-6。

$$\beta_0^*, \beta^* = argmin_{\beta_0,\beta}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - (x_i * \beta + \beta_0))^2\right)$$

公式 4-6 OLS 对应的最小化问题

符号 argmin 是指“使表达式最小的参数”，加和是在行上进行，其中一行包括属性值以及对应的标签。$()^2$ 中的表达式是 y_i 以及用于近似 y_i 的线性函数值之间的错误。对于 2013 年购书方面的花费预测，加和中的表达式对应于结果值减去通过公式 4-4 计算得到的预测值。

公式 4-6 可以用如下语言描述：向量 beta 是以及常量 beta 零星是使期望预测的均方错误最小的值，期望预测的均方错误是指在所有数据行（i=1,...,n）上计算 y_i 与预测生成 y_i 之间的错误平方的平均。公式 4-5 的最小化生成了回归模型的最小均方误差值。该机器学习模型对应于一组实数，即向量 β^* 以及标量 β_0^* 中的数字。

4.2.2 向 OLS 公式中添加一个系数惩罚项

惩罚线性回归问题的数学声明非常类似于公式 4-5。第 3 章介绍岭回归时给出了惩罚线性回归的一个例子。岭回归向公式 4-5 的普通最小二乘法添加了一个惩罚项。岭回归的惩罚项如公式 4-7 所示。

$$\frac{\lambda\beta^T\beta}{2}=\frac{\lambda(\beta_1^2+\beta_2^2+\cdots+\beta_n^2)}{2}$$

公式 4-7 应用于系数（beta）的惩罚项

公式 4-6 的 OLS 问题是选择能最小化均方误差的 β。惩罚回归问题（即公式 4-7）向公式 4-6 的右侧添加系数惩罚项。最小化会考虑尽量平衡均方误差以及系数平方值，而这两方面的目标往往是相互矛盾的。只考虑系数本身很容易最小化系数平方：令每个系数都等于 0。但那样会导致较大的预测错误。类似地，OLS 的解可以使预测错误最小化，但可能导致系数惩罚项变大，取决于 λ 的值有多大。

为什么这样做是有意义的？为了加深理解，回想一下第 3 章的子集选择过程。子集选择通过丢弃一些属性来消除过拟合，实际等同于将这些属性的对应系数设为 0。惩罚回归做同样的事情，但与子集选择直接将一些属性系数设为 0 不同，惩罚线性回归将每个属性系数都减少一些。下面一些例子将帮助对该方法进行可视化。

参数 λ 的取值范围为 0 ～ ∞。如果 λ=0，惩罚项就消失了，问题变为普通最小二乘问题。如果 $\lambda\to\infty$，关于 β 的惩罚就变得非常严格，会使 β 就趋近于 0。（注意到 β_0 并没有包含在惩罚项中，所以预测变为一个常数值，与输入 x 无关）。

正如第 3 章例子所示，岭惩罚项也可以将一些属性排除在外。这个过程是通过为惩罚版的最小化问题生成一族解来实现的，即对不同的 λ 值都求解一个惩罚最小化问题。每一个解都在样本外数据上进行测试，能够最小化样本外错误的解作为最终解，用于后续预测。第 3 章介绍了使用岭回归的一系列步骤。

4.2.3 其他有用的系数惩罚项：Manhattan 以及 ElasticNet

岭惩罚项对于惩罚回归来说并不是唯一有用的惩罚项。任何关于向量长度的指标都可以。使用不同的长度指标可以改变解的重要性。岭回归应用欧式几何的指标（即 β 的平方

和）。另外一个有用的算法称作套索（Lasso）回归，该回归源于出租车的几何路径被称作曼哈顿距离或者 L1 正则化（即 β 的绝对值的和）。套索回归的一些属性非常有用。

岭回归以及套索回归的差异在于对 β（即线性系数向量）的惩罚上。岭回归使用欧式距离的平方，即 β 元素的平方和进行惩罚。套索使用 β 元素绝对值的加和：称作出租车或者曼哈顿距离。岭惩罚项是连接原点与空间点 β 的直线的长度平方。套索惩罚项类似于出租车在城市穿行需要正南正北或者正西正东的移动这样产生的距离。套索惩罚项的公式如下。

$$\lambda\|\beta_1\|=\lambda(|\beta_1|+|\beta_2|+\cdots+|\beta_n|)$$

公式 4-8 基于曼哈顿距离的惩罚项的公式

两条垂直线称作正则线。它们用于定义向量或者操作符的维度。正则线的脚标 1 称作 l_1 正则化，对应绝对值的和。常使用大写的 L1 来标记。脚标为 2 的正则线对应平方和的开方根——欧式距离。不同的系数惩罚函数在生成解时存在一些重要差异。一个主要差异是：套索的系数向量 β^* 是稀疏的，意味着对于不同的 λ 值，许多系数等于 0。相比之下，岭回归 β^* 向量值是密集的，大部分不等于 0。

4.2.4 为什么套索惩罚会导致稀疏的系数向量

稀疏性与系数惩罚函数的直接关系如图 4-1 和图 4-2 所示。这些图是针对含有 2 个属性 $x1$ 与 $x2$ 的问题。

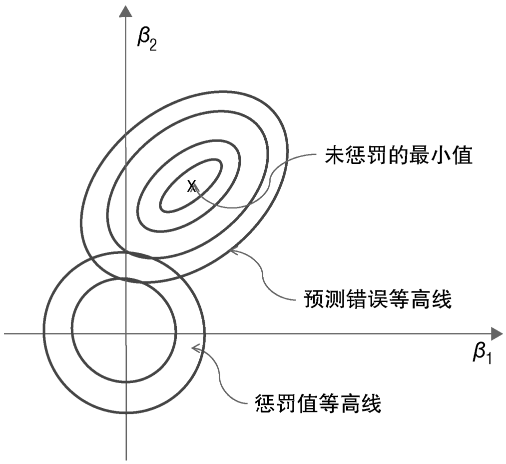


图 4-1 使用平方系数和惩罚项的最优解

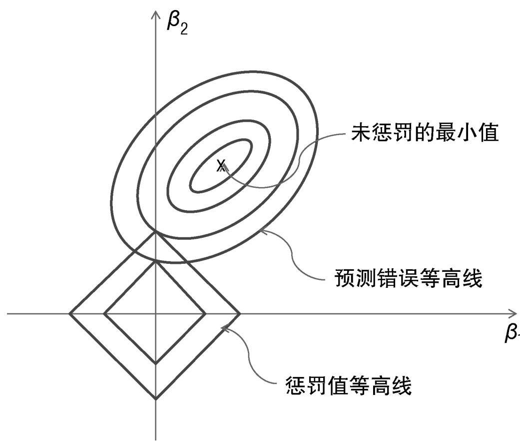


图 4-2 使用系数绝对值和的惩罚项的最优解

图 4-1 与图 4-2 包含 2 组曲线集合。一组曲线集合是同心椭圆，代表公式 4-6 中的最

小均方误差。椭圆代表等高线，线上的均方和为常数。可以将这些椭圆想象为地面上呈椭圆凹陷的地图。对于更接近于中心的椭圆，错误会变小，正如凹陷的海拔高度会随凹陷方向变小。凹陷的最小点使用 x 进行标记。在没有系数惩罚项时,该点对应于最小二乘法的解。

图 4-1 及图 4-2 的曲线集合代表公式 4-7 和公式 4-9 的系数惩罚项，分别是岭惩罚项和套索惩罚项。在图 4-1 中，代表系数惩罚项的曲线是以原点为中心的圆。平方和等于常量，限定了 β_1 以及 β_2 是位于圆上的点。常数惩罚项的曲线形状由使用的距离性质决定：基于平方和的惩罚项对应于圆（称作超球面或者高维空间的 l_2 球面），基于绝对值和的惩罚项对应于菱形（或者 l_1 球面）。小的圆（或者菱形）对应于较小的距离函数。形状虽然由惩罚函数的性质决定，但是每条曲线关联的值由非负参数 λ 来决定。假设图 4-1 中的 2 条曲线分别对应平方和（β_1、β_2）为 1.0 和 2.0 的情况。如果 λ =1，那么和 2 个圆圈关联的惩罚项为 1 和 2。如果 λ =10，那么关联的惩罚项为 10 和 20。图 4-2 中的菱形结果也相同。增加 λ 也会增加与图 4-2 相关的惩罚项。

椭圆环（对应于预测错误平方）离不受限的最小值（图中的 x 标记）越远，椭圆环也变得越大。正如公式 4-6 所示，最小化这 2 个函数的和对应于在预测错误最小化以及系数惩罚最小化之间寻求一种平衡。较大的 λ 值会更多地考虑最小化惩罚项（所有系数为 0）。较小的 λ 值会使得最小值接近于不受限的最小预测错误（图 4-1 及图 4-2 的 x）。

这就是系数平方和的惩罚项与绝对值和的惩罚项之间的区别。公式 4-6 以及公式 4-8 的最小值往往会落在惩罚常数曲线与预测错误曲线的切点上。图 4-1 与图 4-2 为相切的 2 个例子。重要一点是在图 4-1 中，随着 λ 变化以及最小点的移动，平方惩罚项产生的切点一般不会落在坐标轴上。β_1 与 β_2 都不为 0。相比之下，在图 4-2 中，绝对值和惩罚项产生的切点落在了 β_2 的轴上。在 β_2 轴上，β_1=0。

一个稀疏的系数向量相当于算法告诉你可以忽略一些因变量。当 λ 足够小时，β_2 与 β_1 的最优值会远离 β_2 轴，这 2 个值都是非 0 值。较小的惩罚项会使得 β_1 不等于 0，给出了 β_2 与 β_1 的顺序。从某种意义上讲，β_2 要比 β_1 重要，因为随着 λ 变大，β_2 的值不等于 0。回想一下这些系数会乘以属性。如果对应于属性的系数为 0，算法告诉你该属性的重要性要差于非 0 的属性。通过从大到小遍历 λ 值，就可以根据重要性对属性进行排序。下一节会通过一个具体例子来说明，并提供 Python 代码比较属性的重要性，而属性重要性是公式 4-8 求解的一部分。

4.2.5　ElasticNet 惩罚项包含套索惩罚项以及岭惩罚项

在了解如何计算这些系数之前，需要知道惩罚回归问题的泛化定义，即 ElasticNet 形式化。惩罚回归问题的 ElasticNet 形式是使用一个可调节的岭回归以及套索回归的混合。ElasticNet 引入一个额外参数 α 用于控制岭惩罚项以及套索惩罚项的比例。α =1 表示只使

用套索惩罚，不使用岭惩罚。使用 ElasticNet 形式，在求解线性模型系数之前，λ 以及 α 必须提前确定。一般来讲，确定 λ 以及 α 参数的方法是先确定 α 值，然后尝试使用不同的 λ 值。会在后续看到这样计算的原因。

在许多情况下，对 α =1、α =0 或者一些中间值的 α，算法性能差异并不大。但有时差异会很明显，这就需要选择不同的 α 值来确保不必要的性能牺牲。

4.3 求解惩罚线性回归问题

在前面章节中，我们看到求解惩罚线性回归模型等价于求解一个优化问题。有大量通用的数值优化算法可以求解公式 4-6、公式 4-8 以及公式 4-11 对应的优化问题，但是惩罚线性回归问题的重要性促使研究人员开发专用算法，从而能够非常快地生成解。本章将对这些算法进行介绍并且运行相关代码，这样可以帮助理解每种算法的运行机制。本章将介绍 2 种算法：最小角度回归 LARS 以及 Glmnet。之所以选择这 2 种算法是因为它们之间相互关联，并且和前面介绍的方法，如岭回归以及前向逐步回归密切相关。此外，它们训练速度都非常快，并且 Python 包中已经有相关实现。第 5 章会使用 Python 包来引入这些算法求解样例问题。

4.3.1 理解最小角度回归与前向逐步回归的关系

一种非常快速聪明的算法是最小角度回归（LARS）算法，该算法由 Bradley Efron, Trevor Hastie、Iain Johnstone、Robert Tibshirani 等人发明（http://en.wikipedia.org/wiki/Least-angle_regression）。LARS 算法可以理解为一种改进的前向逐步回归算法。

1．前向逐步回归算法

◆ 将 β 的所有值初始化为 0。

在每一步中

◆ 使用已经选择的变量找到残差值。

◆ 确定哪个未使用的变量能够最佳的解释残差，将该变量加入选择变量中。

LARS 算法与前向逐步回归算法非常类似。LARS 与前向逐步回归算法的主要差异是 LARS 在引入新属性时只是部分引入，引入属性过程并非不可逆。LARS 算法过程如下。

2．最小角度回归算法（LARS）

◆ 将 β 的所有值都初始化为 0。

在每一步中

◆ 决定哪个属性与残差有最大的关联。

◆ 如果关联为正，小幅度增加关联系数；关联为负，小幅度减少关联系数。

LARS 算法求解的问题与之前问题稍微不同。然而，它生成的解与套索基本相同，即使结果存在差异，差异也相对较小。之所以介绍 LARS 算法是因为该算法非常接近于套索以及前向逐步回归,LARS 算法很容易理解并且实现起来相对紧凑。通过研究 LARS 的代码，你会理解针对更一般的 ElasticNet 回归求解的具体过程，并且会了解惩罚回归求解的细节。实现 LARS 算法的代码如代码清单 4-1 所示。

代码清单 4-1　用于预测红酒口感的 LARS 算法 -larsWine2.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import datasets, linear_model
from math import sqrt
import matplotlib.pyplot as plot
#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learningdatabases/"
"wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)
```

```
#Normalize columns in x and labels

nrows = len(xList)
ncols = len(xList[0])

#calculate means and variances
xMeans = []
xSD = []
for i in range(ncols):
    col = [xList[j][i] for j in range(nrows)]
    mean = sum(col)/nrows
    xMeans.append(mean)
    colDiff = [(xList[j][i] - mean) for j in range(nrows)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrows)])
    stdDev = sqrt(sumSq/nrows)
    xSD.append(stdDev)

#use calculate mean and standard deviation to normalize xList
xNormalized = []
for i in range(nrows):
    rowNormalized = [(xList[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncols)]
    xNormalized.append(rowNormalized)

#Normalize labels
meanLabel = sum(labels)/nrows
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] -
    meanLabel) for i in range(nrows)])/nrows)

labelNormalized = [(labels[i] - meanLabel)/sdLabel \
    for i in range(nrows)]

#initialize a vector of coefficients beta
beta = [0.0] * ncols

#initialize matrix of betas at each step
betaMat = []
betaMat.append(list(beta))

#number of steps to take
```

```
nSteps = 350
stepSize = 0.004

for i in range(nSteps):
    #calculate residuals
    residuals = [0.0] * nrows
    for j in range(nrows):
        labelsHat = sum([xNormalized[j][k] * beta[k]
            for k in range(ncols)])
        residuals[j] = labelNormalized[j] - labelsHat

    #calculate correlation between attribute columns from
    #normalized wine and residual
    corr = [0.0] * ncols

    for j in range(ncols):
        corr[j] = sum([xNormalized[k][j] * residuals[k] \
            for k in range(nrows)]) / nrows

    iStar = 0
    corrStar = corr[0]

    for j in range(1, (ncols)):
        if abs(corrStar) < abs(corr[j]):
            iStar = j; corrStar = corr[j]

    beta[iStar] += stepSize * corrStar / abs(corrStar)
    betaMat.append(list(beta))


for i in range(ncols):
    #plot range of beta values for each attribute
    coefCurve = [betaMat[k][i] for k in range(nSteps)]
    xaxis = range(nSteps)
    plot.plot(xaxis, coefCurve)

plot.xlabel("Steps Taken")
plot.ylabel(("Coefficient Values"))
plot.show()
```

代码主要包括 3 个部分，这里先简单介绍，后续会详细说明。

（1）读入数据以及列名称，将数据转换为属性列表（Python 的 list 对象，每个元素对应一个样例，该样例使用包含属性的 list 对象表示）以及标签。

（2）对属性以及标签进行归一化。

（3）对问题进行求解获得结果 β^* β_0 *。

第一段代码用于读入整个文件，将文件头分离出来，使用符号“;”获得文件头分隔的属性名列表，将剩下的行切分为浮点数列表，将所有记录封装为列表（一行记录）的列表，将标签封装为列表。对这些数据结构使用 Python 的列表（list）类型，是因为算法要对行和列进行遍历，而 Pandas 的数据帧类型对这个目的来说太慢。

第 2 段代码使用第 2 章的归一化方法。在第 2 章中，属性归一化的目的是将属性值转换为同等尺度，从而可以方便地绘制以及充分填充坐标。在惩罚线性回归中，归一化一般作为第一步，原因大致相同。

LARS 算法的每一步都会固定增加 β 变量值。属性尺度不同，固定增量对不同属性的影响也不同。同样，如果改变一个属性的尺度（从英里变为英尺），答案会显著不同。因为这些原因，惩罚线性回归包一般使用第 2 章中的加权方法，即归约到 0 均值（减去平均值）、单位方差（用标准方差去除）。算法包一般也会提供非归一化的选项，但是很少听说不进行归一化有什么优势。

第 3 节以及最后一节代码用于求解 β^*、β_0^*。因为算法运行在归一化的变量上，因此不需要截距 β_0 *。截距 β_0^* 一般用于表示标签值与加权属性值之间的差异。因为所有属性已经被归约到 0 均值，所以标签与加权属性的期望没有偏差，β_0 * 就不再需要了。注意到有 2 个 beta 相关的列表被初始化。一个称作 beta，其中元素个数与属性个数相同。另一个是类似于矩阵 - 列表的列表 - 用于存储 LARS 算法每一步生成的 beta 列表。这些都涉及了惩罚线性回归以及现代机器学习算法的重要概念。

4.3.2　LARS 如何生成数百个不同复杂度的模型

一般现代的机器学习算法，尤其是惩罚线性回归方法会生成多组解，不仅是单个解。回顾公式 4-6、公式 4-8 以及公式 4-11。公式的左侧是 β-，右侧是通过数据求解得到的数值。在公式 4-6 以及公式 4-8 中，参数 λ 需要通过其他方式确定。正如这些公式中所指定的，当 λ=0 时，问题变为最小二乘法，当 $\lambda \rightarrow \infty$，$\beta^* \rightarrow 0$。所以 β 值取决于公式 4-6、公式 4-8 以及公式 4-11 中的参数 λ。

LARS 算法并不会显式处理 λ，但会有同样效果。LARS 算法从 β=0 开始，只要 β 中的某一个系数能够最大程度地减少错误，那么该系数对应增加。被添加的增量会提升 β 整体的绝对值和。如果增量较小并且被用在最佳属性上，该过程就具有求解公式 4-8 对应的

最小化问题的效果。可以在代码清单 4-1 中追踪该过程。

基本迭代过程只包含几行代码，从 for 循环开始迭代 n 步。迭代的起始点是 β 的一个值。第一遍，所有值被设为 0。后续遍历，使用上一次迭代的结果。每次迭代包括 2 步。首先，使用 β 计算残差，残差是指观测结果与预测结果的差异。在该例中，预测结果值等于属性值乘以系数然后加和。第 2 步找到每个属性与残差的关联，从而决定哪个属性对降低残差贡献最大。2 个变量的关联值等于归一化（减去均值后除以标准差）值的乘积。

如果 2 个变量完全相关，比如一个变量是另一个变量的缩放版，那以正的缩放对应于正关联，负的缩放对应于负关联。如果 2 个变量相互独立，那么他们的关联系数为 0。维基百科对关联的解释（http://en.wikipedia.org/wiki/Correlation_and_dependence）展示变量相互关联的例子。列表 corr 包含每个属性的计算结果。你可能注意到严格来讲，代码忽略了对均值、残差以及归一化属性的标准差的计算。这里之所以还有效是因为属性已经被提前归一化为标准差为 1 的值，并且因为这些值被用于寻找最大关联，将所有值乘以一个常数不会影响变量顺序。

一旦关联计算完成，决定哪个属性与残差有最大关联（绝对值最大）就变得简单。β 列表中的对应元素会增加一点。如果关联为正，增量就为正，否则增量为负。之后利用 β 的新值再进行迭代计算。

LARS 算法生成的结果如图 4-3 中的曲线所示。对图进行理解可以按照如下的方法：沿着迭代方向想象有一个点，在该点上，一条垂直线会穿过所有系数曲线。垂直线与系数曲线相交的值是 LARS 算法在该步得到的系数。如果生成曲线需要 350 步，对应就会有 350 个系数集合。每个集合是针对特定 λ、对公式 4-8 进行优化获得的结果。这也产生了一个问题：就是使用哪个集合是最优的。该问题会在稍后进行解答。

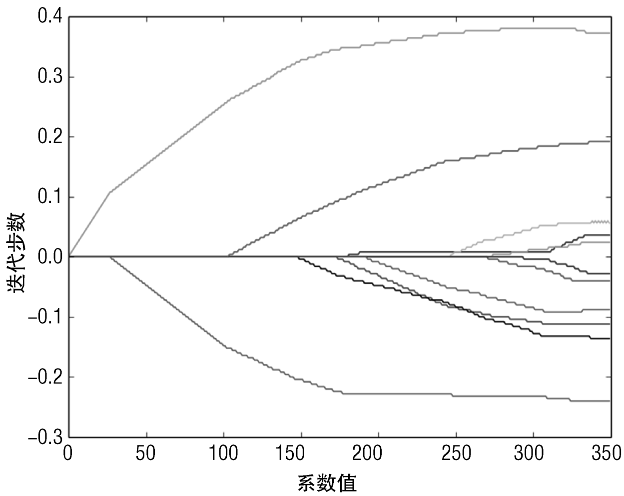


图 4-3　在红酒数据上 LARS 回归的系数曲线

注意到对于前 25 步，只有一个系数值非 0。这对应于套索回归的稀疏属性。第一个不等于 0 的属性是酒精，在此后的一段时间内，这是 LARS 回归唯一使用的变量。随着迭代步数的增加，第 2 个变量开始出现，这样的过程一直持续到所有变量被包含到解中。系数远离 0 的过程可以看作是变量重要性的反映。在某些情况下，如果要去掉一些变量的预测，就应该尽量去掉后面出现的而不是开始出现的变量。

重点中的重点

生成变量重要性排序是惩罚线性回归模型的一个重要特征。这使得该模型成为模型初步研究中的一个便捷的工具，因为它们将帮助构建关于保留哪个变量、丢弃哪个变量的决定，这个过程也称作特征工程。你会在后续看到树集成同样会产生变量重要性的度量。不是所有的机器学习方法都会生成此类信息。虽然你总是可以通过尝试所有组合，如 1 个变量、2 个变量等等来生成关于特征的重要性信息，但即使对只有 10 个属性的红酒数据集来说，遍历所有可能子集也需要 10 的阶乘这么多次的训练，这对算法来讲是不可实现的。

4.3.3　从数百个 LARS 生成结果中选择最佳模型

目前关于利用红酒的化学属性上来预测红酒口感得分，已经有 350 个可能的解。如何选择最佳的解？为了选择使用的曲线，你需要了解 350 个选择中每种选择的优劣。正如第 3 章所讨论的，性能是指在样本外数据上的性能。第 3 章列出了几种在预留数据上评估性能的方法。代码清单 4-2 展示了执行 10 折交叉验证的代码来决定部署使用的最佳系数集合。

10 折交叉验证是将输入数据切分为 10 份几乎均等的数据，将其中一份数据移除，使用剩下数据进行训练，然后再在移除的数据上进行测试。通过遍历 10 份数据，每次移除一份数据用于测试，从而可以对错误进行估计从而估计预测的变化情况。

代码清单 4-2　利用 10 折交叉验证来确定最佳系数集合—larsWineCV.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import datasets, linear_model
from math import sqrt
import matplotlib.pyplot as plot

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
```

```
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)
#Normalize columns in x and labels

nrows = len(xList)
ncols = len(xList[0])

#calculate means and variances
xMeans = []
xSD = []
for i in range(ncols):
    col = [xList[j][i] for j in range(nrows)]
    mean = sum(col)/nrows
    xMeans.append(mean)
    colDiff = [(xList[j][i] - mean) for j in range(nrows)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrows)])
    stdDev = sqrt(sumSq/nrows)
    xSD.append(stdDev)

#use calculated mean and standard deviation to normalize xList
xNormalized = []
for i in range(nrows):
```

```
    rowNormalized = [(xList[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncols)]
    xNormalized.append(rowNormalized)

#Normalize labels
meanLabel = sum(labels)/nrows
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] - meanLabel)
    for i in range(nrows)])/nrows)

labelNormalized = [(labels[i] - meanLabel)/sdLabel \
    for i in range(nrows)]

#Build cross-validation loop to determine best coefficient values.

#number of cross-validation folds
nxval = 10

#number of steps and step size
nSteps = 350
stepSize = 0.004

#initialize list for storing errors.
errors = []
for i in range(nSteps):
    b = []
    errors.append(b)


for ixval in range(nxval):
    #Define test and training index sets
idxTest = [a for a in range(nrows) if a%nxval == ixval*nxval]
idxTrain = [a for a in range(nrows) if a%nxval != ixval*nxval]

#Define test and training attribute and label sets
xTrain = [xNormalized[r] for r in idxTrain]
xTest = [xNormalized[r] for r in idxTest]
labelTrain = [labelNormalized[r] for r in idxTrain]
labelTest = [labelNormalized[r] for r in idxTest]

#Train LARS regression on Training Data
```

```
nrowsTrain = len(idxTrain)
nrowsTest = len(idxTest)

#initialize a vector of coefficients beta
beta = [0.0] * ncols

#initialize matrix of betas at each step
betaMat = []
betaMat.append(list(beta))

for iStep in range(nSteps):
    #calculate residuals
    residuals = [0.0] * nrows
    for j in range(nrowsTrain):
        labelsHat = sum([xTrain[j][k] * beta[k]
            for k in range(ncols)])
        residuals[j] = labelTrain[j] - labelsHat

    #calculate correlation between attribute columns
    #from normalized wine and residual
    corr = [0.0] * ncols

    for j in range(ncols):
        corr[j] = sum([xTrain[k][j] * residuals[k] \
            for k in range(nrowsTrain)]) / nrowsTrain

    iStar = 0
    corrStar = corr[0]

    for j in range(1, (ncols)):
        if abs(corrStar) < abs(corr[j]):
            iStar = j; corrStar = corr[j]

    beta[iStar] += stepSize * corrStar / abs(corrStar)
    betaMat.append(list(beta))

    #Use beta just calculated to predict and accumulate out of
    #sample error - not being used in the calc of beta
    for j in range(nrowsTest):
        labelsHat = sum([xTest[j][k] * beta[k] for k in range
```

```
                (ncols)])
            err = labelTest[j] - labelsHat
            errors[iStep].append(err)

cvCurve = []
for errVect in errors:
    mse = sum([x*x for x in errVect])/len(errVect)
    cvCurve.append(mse)

minMse = min(cvCurve)
minPt = [i for i in range(len(cvCurve)) if cvCurve[i] == minMse ][0]
print("Minimum Mean Square Error", minMse)
print("Index of Minimum Mean Square Error", minPt)

xaxis = range(len(cvCurve))
plot.plot(xaxis, cvCurve)

plot.xlabel("Steps Taken")
plot.ylabel(("Mean Square Error"))
plot.show()

Printed Output:
('Minimum Mean Square Error', 0.5873018933136459)
('Index of Minimum Mean Square Error', 311)
```

利用交叉验证进行模型选择

代码清单 4-2 类似于代码清单 4-1。在交叉验证循环中（循环 nxval 次），这种差异逐渐变得清晰。在本例中，nxval=10，当然，nxval 可以设置为其他值。关于交叉验证份数的选择，如果份数较少，那么每次训练使用的数据也较少。如果设为 5 折，那么每次训练时会预留 20% 的数据。如果使用 10 折，只会留出 10% 的数据。正如第 3 章所看到的，在较少的数据上训练会降低算法的准确性。然而，切分份数过多也意味着在训练过程中需要多次遍历，这会显著增加训练时间。

在进行交叉验证循环前，首先初始化一个错误列表。该错误列表包含 LARS 算法中每一步迭代的错误。算法会对所有 10 折交叉验证的每折错误进行累加。在交叉验证循环中，你会看到训练集和测试集的生成过程。这里使用一个取模函数来定义这些集合。在有些情况下，需要使用分层抽样。例如要在一个非平衡数据集上构建分类器，其中某一类的样本点很少。如果希望训练集能够代表整个数据集，那么需要按照类来抽样数据，从而确保样

本内和样本外的类大小比例一致。

你可能倾向于使用一个随机函数来定义训练集和测试集，但是要清楚数据集中的任何模式都可能影响抽样过程（即如果观测不是可交换的）。例如，如果数据是在每周的工作日进行采集，那么使用 5 折交叉验证的取余函数可能导致一个集合包含所有周一，另一个集合包含所有周二，等等。

在交叉验证的每一份数据上累加错误以及评估结果

一旦训练集和测试集定义好，LARS 算法的迭代就可以开始。该过程和代码清单 4-1 类似，但也存在一些区别：首先，算法的基本迭代在训练集上展开，而不是在所有数据集上展开；其次，对于每次迭代，每份数据上的测试错误使用 \beta 当前值、数据属性以及数据标签计算得到，对应代码在交叉验证循环的最底端。β 每次更新时，在测试集上重复上述过程，将错误累加到“error”列表中，此后对每个列表进行平方取平均就变得简单。代码最终会生成一条曲线，曲线上的每个点与每次迭代的均方误差（MSE）值相对应，MSE 是在 10 份数据上得到的 MSE 的平均值。

你可能会担心测试数据是否被合理使用。一定要防止把测试数据引入训练过程，有多种方式可能会违背该限制。最主要的是，要确保测试数据不能在计算 β 的增量时使用，β 的增量计算只能使用训练数据。

关于模型选择以及训练次序的实际考虑

MSE 曲线与 LARS 迭代步数的关系如图 4-4 所示。该曲线展示了一个非常普遍的模式。基本上 MSE 随着迭代步数的增加而单调递减。严格来讲，从程序的输出来看，最小

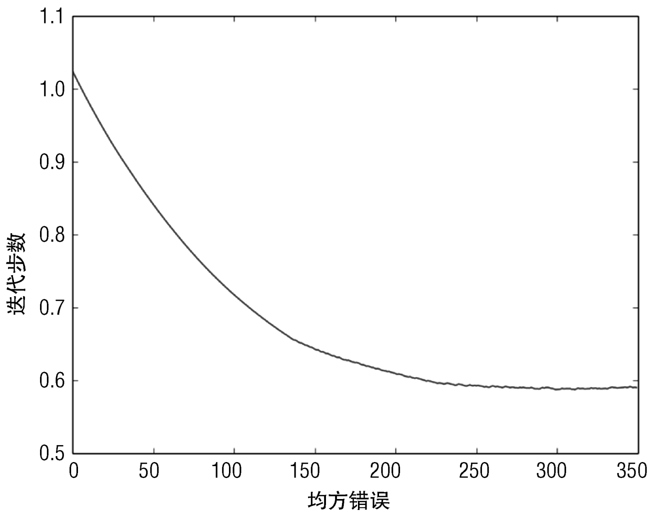


图 4-4 在红酒数据上 LARS 算法的均方误差（基于交叉验证的错误评估）

值在第 311 步左右。但是该图显示最小值比较稳定,相对周围没有明显突出。在某些情况下,该曲线会在某个点上到达一个明显的最小值，往左或者往右都会显著增加。使用交叉验证来决定 350 个解中，LARS 生成的哪个解应该用于构建预测。在本例中，最小值在第 311 步出现，β 的第 311 个系数集合可以用于实际部署。关于部署时具体使用哪个解，一般倾向于使用较为保守的方案。对于惩罚回归来讲，保守的解意味着系数值较小的解。按照惯例，对样本外性能，简单模型一般在左边，复杂模型一般在右边。简单模型往往有更好的泛化性能，即它们对新数据的预测更为准确。更保守的模型在样本外数据上的性能一般出现在图的左边。

代码中的 LARS 算法以及交叉验证首先遍历整个数据集，然后运行交叉验证。实际上，你很可能先运行交叉验证，然后在整个数据集上训练模型。交叉验证的目的是估计模型能够到达的 MSE 性能值,并了解数据集能够支持学习怎样复杂度的模型。如果你能回忆起来,第 3 章讨论过数据集大小以及模型复杂度的问题。交叉验证（或者其他基于预留数据进行性能评估）用来确定部署模型的复杂度。注意是复杂度而不是特定模型（即并不是 β 的特定取值）。正如在代码清单 4-2 中所看到的，使用 10 折交叉验证，实际训练了 10 个模型，在 10 个模型中，无法决定哪个是最佳模型。好的经验是在完整数据集上训练，使用交叉验证确定部署哪个模型。对于代码清单 4-2 中的例子，交叉验证在训练阶段的第 311 步取到 MSE 的最小值 0.59。图 4-5 的系数曲线是在整个数据集上训练得到的。因为并不了解 350 个系数集合的哪一个（见图 4-5）应该被部署使用，所以使用交叉验证确定。交叉验证会生成对 MSE 的合理估计，告诉我们将第 311 个模型从训练集部署到整个数据集上。

4.3.4 使用 Glmnet：非常快速并且通用

glmnet 算法由 Jerome Friedman 教授以及他的同事们在斯坦福大学开发。glmnet 算法解决公式 4-11 给出的 ElasticNet 问题。回忆一下 ElasticNet 算法引入对惩罚函数的泛化，包括套索惩罚（绝对值加和）以及岭惩罚（平方和）。ElasticNet 通过参数 λ 来决定系数惩罚项相对于拟合错误的重要程度。算法同时包括参数 α，该参数用于决定惩罚项与岭回归（α=0）以及套索回归（α=1）的接近程度。glmnet 算法生成了完整的系数曲线，类似于 LARS 算法。只要 LARS 算法将系数累加量加入 β，就能使曲线前进，glmnet 算法会稳定减少 λ 来推进系数曲线。公式 4-9 为 Friedman 论文中的关键公式：求解公式 11 中的权重系数的迭代公式——ElasticNet 公式。

$$\beta_j^{\sim} \leftarrow \frac{S\left(\frac{1}{m}\sum_{i=1}^{m} x_{ij} r_i + \beta_j^{\sim}, \lambda\alpha\right)}{1+\lambda(1-\alpha)}$$

公式 4-9 glmnet 算法按照坐标的更新

公式 4-9 对应于 Friedman 论文中的公式 5 与公式 8 的组合（假设你们对数学感兴趣）。看上去比较复杂，但是仔细观察会发现与上一节的 LARS 方法存在一定的关系和相似性。

Glmnet 与 LARS 算法工作原理的比较

公式 4-9 为 β 更新的基本公式。LARS 更新公式是找到与残差关联最大的属性并小幅增加(减少)对应系数。修改后的公式 4-9 稍微麻烦一些。它使用一个箭头而不是一个等号。箭头表示“被映射到”。注意到 β_j~出现在箭头的两边。箭头右侧是老的 β_j~值，箭头左边(箭头所指方向）是 β_j~的新值。经过若干次遍历（对应于公式 4-12 中的迭代），β_j~停止改变，此时算法针对给定的 λ 以及 α 收敛到了最终解。现在是时候讨论一下系数曲线了。

首先应该注意的是加和中的 $x_{ij}r_i$ 项。$x_{ij}r_i$ 在 i 上相加（即在数据的行上计算）生成了第 j 个属性与残差的关联。回忆一下 LARS 回归算法的每一步，计算每个属性与残差的关联。在 LARS 算法中，通过考虑所有关联值来确定哪个属性与残差的关联最大，增加关联最大的属性系数。使用 glmnet 算法，关联计算稍微不同。

使用 glmnet，与残差的关联用于计算系数的变化幅度。但是在 β_j~改变前，需要遍历函数 S()。函数 S() 是套索系数缩减函数。该函数在图 4-5 中绘制。正如你在图 4-5 看到的，如果第 1 个输入小于第 2 个输入，则输出为 0。如果第一个输入大于第 2 个输入，则输出变为第 1 个输入减去第 2 维度上的输入。这个称作软限制。

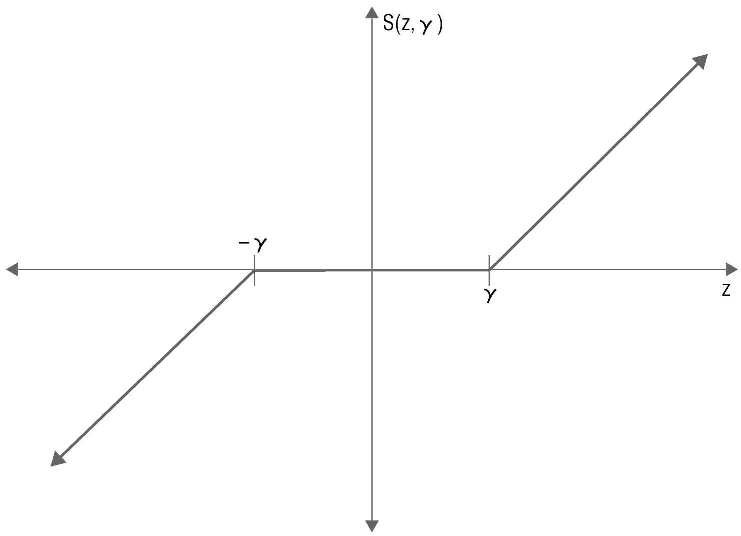


图 4-5 S() 函数图

代码清单 4-3 为 glmnet 算法的代码。可以看到用于更新 β 的公式 4-12 是如何用来生

成 ElasticNet 系数曲线的。代码清单 4-3 中的公式编号与 Friedman 论文一致。论文可以在网上找到，如果感兴趣可以查阅更多的数学细节。

初始化并且进行 Glmnet 算法的迭代

迭代从一个较大的 λ 值开始。λ 足够大使得所有 β 等于 0。公式 4-9 给出如何计算 λ 的起始点。对于公式 4-12 的函数 S()，如果第 1 个输入（$x_{ij}r_i$ 的关联）小于第 2 个输入 $-\lambda\alpha$，则函数输出为 0。迭代从 β 都等于 0 开始，此时残差等于标签值。决定 lambda 初始值的流程为：首先计算每个属性与标签的关联，找到关联值最大的属性，使最大关联刚好等于 $\lambda\alpha$ 来求解 λ 的值。这就是 λ 的最大值，该值导致 β 的所有值都等于 0。

接着迭代从减少 λ 开始，使用稍微小于 1 的系数乘以 λ 来完成。Friedman 建议乘子最好满足 λ^100 = 0.001，此时乘子大约等于 0.93。如果算法运行很长时间都没有收敛，那么增加 λ 的乘子使其接近于 1。Friedman 的代码实现将步长从 100 增加到 200，经过 200 步使得起始点 λ 从 1 降为 0.001。在代码清单 4-3 中，可以直接控制乘子大小。系数曲线如图 4-8 所示。

代码清单 4-3 Glmnet 算法 -glmnetWine.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import datasets, linear_model
from math import sqrt
import matplotlib.pyplot as plot
def S(z, gamma):
    if gamma >= abs(z):
        return 0.0
    return (z/abs(z))*(abs(z) - gamma)

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
```

```
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

#Normalize columns in x and labels

nrows = len(xList)
ncols = len(xList[0])

#calculate means and variances
xMeans = []
xSD = []
for i in range(ncols):
    col = [xList[j][i] for j in range(nrows)]
    mean = sum(col)/nrows
    xMeans.append(mean)
    colDiff = [(xList[j][i] - mean) for j in range(nrows)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrows)])
    stdDev = sqrt(sumSq/nrows)
    xSD.append(stdDev)

#use calculate mean and standard deviation to normalize xList
xNormalized = []
for i in range(nrows):
    rowNormalized = [(xList[i][j] - xMeans[j])/xSD[j]
                     for j in range(ncols)]
    xNormalized.append(rowNormalized)

#Normalize labels
```

```
meanLabel = sum(labels)/nrows
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] -
                meanLabel) for i in range(nrows)])/nrows)

labelNormalized = [(labels[i] - meanLabel)/sdLabel for i in
range(nrows)]

#select value for alpha parameter

alpha = 1.0

#make a pass through the data to determine value of lambda that
# just suppresses all coefficients.
#start with betas all equal to zero.

xy = [0.0]*ncols
for i in range(nrows):
    for j in range(ncols):
        xy[j] += xNormalized[i][j] * labelNormalized[i]

maxXY = 0.0
for i in range(ncols):
    val = abs(xy[i])/nrows
    if val > maxXY:
        maxXY = val

#calculate starting value for lambda
lam = maxXY/alpha

#this value of lambda corresponds to beta = list of 0's
#initialize a vector of coefficients beta
beta = [0.0] * ncols

#initialize matrix of betas at each step
betaMat = []
betaMat.append(list(beta))

#begin iteration
nSteps = 100
lamMult = 0.93 #100 steps gives reduction by factor of 1000 in
```

```
                # lambda (recommended by authors)
nzList = []

for iStep in range(nSteps):
    #make lambda smaller so that some coefficient becomes non-zero
    lam = lam * lamMult

    deltaBeta = 100.0
    eps = 0.01
    iterStep = 0
    betaInner = list(beta)
    while deltaBeta > eps:
        iterStep += 1
        if iterStep > 100: break

        #cycle through attributes and update one-at-a-time
        #record starting value for comparison
        betaStart = list(betaInner)
        for iCol in range(ncols):

            xyj = 0.0
            for i in range(nrows):
                #calculate residual with current value of beta
                labelHat = sum([xNormalized[i][k]*betaInner[k]
                                for k in range(ncols)])
                residual = labelNormalized[i] - labelHat

                xyj += xNormalized[i][iCol] * residual

            uncBeta = xyj/nrows + betaInner[iCol]
            betaInner[iCol] = S(uncBeta, lam * alpha) / (1 +
                                            lam * (1 - alpha))

        sumDiff = sum([abs(betaInner[n] - betaStart[n])
                       for n in range(ncols)])
        sumBeta = sum([abs(betaInner[n]) for n in range(ncols)])
        deltaBeta = sumDiff/sumBeta
    print(iStep, iterStep)
    beta = betaInner
```

```
        #add newly determined beta to list
        betaMat.append(beta)

        #keep track of the order in which the betas become non-zero
        nzBeta = [index for index in range(ncols) if beta[index] != 0.0]
        for q in nzBeta:
            if (q in nzList) == False:
                nzList.append(q)

    #print out the ordered list of betas
    nameList = [names[nzList[i]] for i in range(len(nzList))]
    print(nameList)

    nPts = len(betaMat)
    for i in range(ncols):
        #plot range of beta values for each attribute
        coefCurve = [betaMat[k][i] for k in range(nPts)]
        xaxis = range(nPts)
        plot.plot(xaxis, coefCurve)

    plot.xlabel("Steps Taken")
    plot.ylabel(("Coefficient Values"))
    plot.show()

    #Printed Output:
    #['"alcohol"', '"volatile acidity"', '"sulphates"',
    #'"total sulfur dioxide"', '"chlorides"', '"fixed acidity"', '"pH"',
    #'"free sulfur dioxide"', '"residual sugar"', '"citric acid"',
    #'"density"']
```

图 4-8 为代码清单 4-3 生成的系数曲线。该曲线接近于 LARS 生成的曲线（见图 4-6），类似但不相同。LARS 以及套索常常生成相同曲线，但有时结果不同。决定哪种方法好的唯一方式是在样本外数据集上测试，观察哪种方法性能更好。

套索模型的开发过程与 LARS 算法相同，使用第 3 章的方法在样本外数据上进行测试（如 n 折交叉验证），基于样本外数据上的结果来决定模型的最优复杂度，然后在完整训练集上进行训练来构建系数曲线，从中挑出样本外性能最好的步长对应的解。

本章介绍了惩罚线性回归模型对应的两个最小化问题的求解方法。学习了这两个算法是如何工作的，以及它们之间的关系和实现细节。这些都为后续使用相关的 Python 代码

包奠定基础，也便于理解第 5 章中关于模型的不同扩展方法。

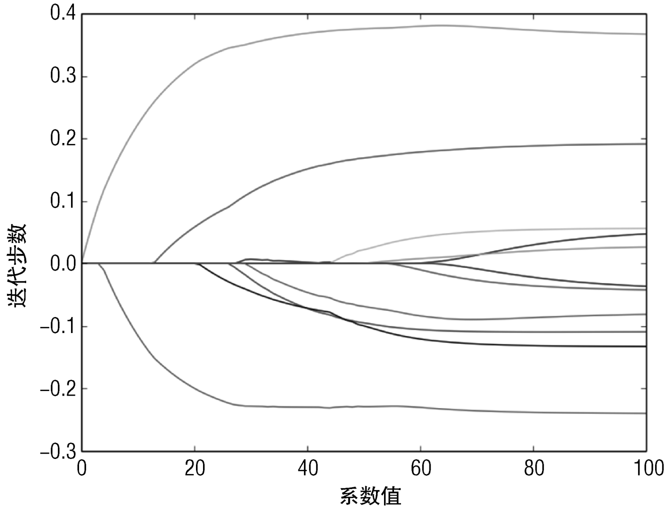


图 4-6 预测红酒口感的 glmnet 模型的系数曲线

4.4 输入为数值型数据的线性回归方法的扩展

目前为止，算法开发关注于回归问题，其中要预测的输出为实数值。如何把讨论的方法推广到分类问题，其中输出为两个离散值（或者更多），比如“点击”或者“未点击”？有多种方法将目前的回归问题推广到分类问题。

4.4.1 使用惩罚回归求解分类问题

对于二分类问题，将二值转换为实数值往往会获得好结果。该过程将 2 个类别值编码为 1 和 0（或者 +1 和 −1）。通过这个简单的安排，标签列表变为一串实数值，之前讨论的算法就可以应用上。虽然有很多扩展方法，但上述方法往往是一个不错的方法。上述简单编码的方法要比复杂方法训练更快，这个特性很重要。

代码清单 4-4 给出了在岩石 - 水雷数据集上将类别属性替换为 0 和 1 的一个例子。回想下第 2 章岩石 - 水雷数据集对应于一个分类问题。数据集来自于一次具体实验，实验目标是确定能否通过声纳来检测未爆炸的水雷。除水雷外，其他物体也会反射声波，那么预测的问题是确定反射声波是来自水雷还是海底岩石。

实验使用的声纳波形称作啁啾波形。啁啾波形在脉冲传输过程中会发生频率升降。岩石水雷数据集中的 60 个属性是在 60 个不同时间点抽样获得的返回脉冲，这些脉冲对应于啁啾脉冲的 60 个不同频段。

代码清单 4-4 展示了如何将分类标签 R 和 M 转换为 0.0 以及 1.0，从而将问题转换为普通回归问题，然后使用 LARS 算法来构建分类器。代码清单 4-4 对整个数据集遍历一遍。正如上一节所讨论的，你会联想到使用交叉验证或者预留数据来选择最优的模型复杂度。我们会在第 5 章对这些设计过程进行回顾并比较它们的性能。这里目标集中在如何把已经提到的回归工具应用到分类问题上。

代码清单 4-4 通过为二类标签赋数值来将普通分类问题转换为普通线性回归问题

```
__author__ = 'mike_bowles'
import urllib2
import sys
from math import sqrt
import matplotlib.pyplot as plot

#read data from uci data repository
target_url = "https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)


#arrange data into list for labels and list of lists for attributes
xList = []


for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

#separate labels from attributes, convert from attributes from
#string to numeric and convert "M" to 1 and "R" to 0

xNum = []
labels = []

for row in xList:
    lastCol = row.pop()
    if lastCol == "M":
        labels.append(1.0)
```

```
    else:
        labels.append(0.0)
    attrRow = [float(elt) for elt in row]
    xNum.append(attrRow)
#number of rows and columns in x matrix
nrow = len(xNum)
ncol = len(xNum[1])


#calculate means and variances
xMeans = []
xSD = []
for i in range(ncol):
    col = [xNum[j][i] for j in range(nrow)]
    mean = sum(col)/nrow
    xMeans.append(mean)
    colDiff = [(xNum[j][i] - mean) for j in range(nrow)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrow)])
    stdDev = sqrt(sumSq/nrow)
    xSD.append(stdDev)

#use calculate mean and standard deviation to normalize xNum
xNormalized = []
for i in range(nrow):
    rowNormalized = [(xNum[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncol)]
    xNormalized.append(rowNormalized)

#Normalize labels
meanLabel = sum(labels)/nrow
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] -
    meanLabel) for i in range(nrow)])/nrow)

labelNormalized = [(labels[i] - meanLabel)/sdLabel for i in range(nrow)]

#initialize a vector of coefficients beta
beta = [0.0] * ncol

#initialize matrix of betas at each step
betaMat = []
```

```
betaMat.append(list(beta))

#number of steps to take
nSteps = 350
stepSize = 0.004
nzList = []

for i in range(nSteps):
    #calculate residuals
    residuals = [0.0] * nrow
    for j in range(nrow):
        labelsHat = sum([xNormalized[j][k] * beta[k]
           for k in range(ncol)])
        residuals[j] = labelNormalized[j] - labelsHat
    #calculate correlation between attribute columns from
    #normalized X and residual
    corr = [0.0] * ncol

    for j in range(ncol):
        corr[j] = sum([xNormalized[k][j] * residuals[k]
            for k in range(nrow)]) / nrow

    iStar = 0
    corrStar = corr[0]

    for j in range(1, (ncol)):
        if abs(corrStar) < abs(corr[j]):
            iStar = j; corrStar = corr[j]

    beta[iStar] += stepSize * corrStar / abs(corrStar)
    betaMat.append(list(beta))


    nzBeta = [index for index in range(ncol) if beta[index] != 0.0]
    for q in nzBeta:
        if (q in nzList) == False:
            nzList.append(q)

#make up names for columns of xNum
names = ['V' + str(i) for i in range(ncol)]
```

```
nameList = [names[nzList[i]] for i in range(len(nzList))]

print(nameList)
for i in range(ncol):
    #plot range of beta values for each attribute
    coefCurve = [betaMat[k][i] for k in range(nSteps)]
    xaxis = range(nSteps)
    plot.plot(xaxis, coefCurve)

plot.xlabel("Steps Taken")
plot.ylabel(("Coefficient Values"))
plot.show()

#Printed Output:
#['V10', 'V48', 'V44', 'V11', 'V35', 'V51', 'V20', 'V3', 'V21', 'V15',
# 'V43', 'V0', 'V22', 'V45', 'V53', 'V27', 'V30', 'V50', 'V58', 'V46',
# 'V56', 'V28', 'V39']
```

图 4-7 为 LARS 算法关联的系数曲线。曲线形状与红酒口感预测曲线类似。然而，这里的曲线数量更多，因为岩石 - 水雷数据集属性更多（岩石水雷数据包含 60 个属性以及 208 行数据）。基于第 3 章的讨论结果，你可能认为最优解不会使用所有属性。第 5 章会看到这种属性取舍的结果（这些结果会显著影响解）以及不同方法的性能优劣。

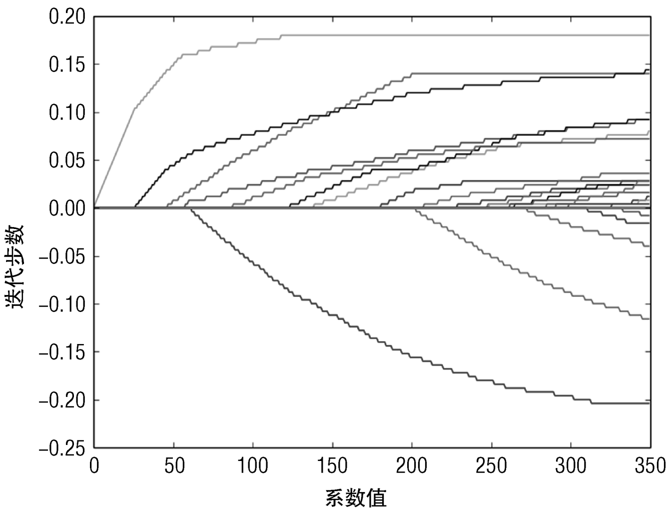


图 4-7 对岩石 − 水雷分类问题，通过转换标签获得的系数曲线

另一类扩展方法是使用输出的似然函数来定义问题，也被称作逻辑回归。glmnet 算法也可以求解逻辑回归，Friedman 的原始论文对 glmnet 逻辑回归进行综述并将其扩展到多类别分类问题，即对超过 2 种离散输出的预测问题。第 5 章将介绍该算法针对二分类以及多分类的版本。

4.4.2　求解超过 2 种输出的分类问题

一些问题需要在多个结果中进行决策。例如，假设你会为你网站的访客呈现多个链接。访客可能会点击链接中的任何一个，点击返回按钮或者完全退出网站。与整数型的红酒口感得分不同，这里存在多种可选方案。红酒口感为 4 的得分很自然地处在 3 和 5 之间，如果改变一个属性（比如酒精含量）使得分从 3 变为 4，改变更多的话会使得分进一步向相同方向移动。而网站访客的行为结果没有这样的顺序。这被称作多分类问题。

你总可以使用一个两类分类器来解决多分类问题。该技术称作一对所有或者一对其余，从名字上大概可以了解算法是怎么工作的。基本上把多分类问题分解为多个二分类问题。例如，你可以预测访问者是否会离开网站或者做其他选择。另一个二分类问题是预测用户是否会点击返回按钮或者点击其他的可用链接。总之能生成多少种相反输出，就可以折腾出多少个二分类问题。二分类器在预测时都会输出具体数值，如代码清单 4-4 中的 LARS 分类器。这些一对其余的分类器中，预测输出最大的作为获胜结果。第 5 章在玻璃数据集上实现了该算法，其中包含 6 种不同的输出结果。

4.4.3　理解基扩展：使用线性方法来解决非线性问题

本质上，线性方法假设分类以及回归预测可以表示为可用属性的线性组合。如果你有理由怀疑线性模型不够的话该怎么办？仍然可以通过基扩展使用线性模型来处理非线性关系。基扩展的基本想法是问题中的非线性可以通过属性的多项式组合来近似（或者属性的其他非线性函数）；你可以向线性回归公式添加原始属性的幂作为回归因子，通过线性方法来确定多项式回归的系数集合。

为了理解为什么该方法有效，可以看代码清单 4-5。代码从红酒口感数据集开始。本章之前提到利用线性模型可以得到酒精是决定口感最重要的属性。这种关系可能不是直线，尤其在酒精度特别高或者特别低的情况下，直线可能弯曲。

代码清单 4-5 展示了如何对上述观点进行验证。

代码清单 4-5　使用基扩展解决红酒口感预测问题

```
__author__ = 'mike-bowles'

import urllib2
```

```
import matplotlib.pyplot as plot
from math import sqrt, cos, log

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)
xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)


#extend the alcohol variable (the last column in that attribute matrix
xExtended = []
alchCol = len(xList[1])


for row in xList:
    newRow = list(row)
    alch = row[alchCol - 1]
    newRow.append((alch - 7) * (alch - 7)/10)
    newRow.append(5 * log(alch - 7))
    newRow.append(cos(alch))
    xExtended.append(newRow)

nrow = len(xList)
```

```
v1 = [xExtended[j][alchCol - 1] for j in range(nrow)]

for i in range(4):
    v2 = [xExtended[j][alchCol - 1 + i] for j in range(nrow)]
    plot.scatter(v1,v2)

plot.xlabel("Alcohol")
plot.ylabel(("Extension Functions of Alcohol"))
plot.show()
```

代码按照之前的方式读入数据。读入数据之后（以及在属性正则化之前），代码对数据行进行扫描，向数据行中添加一些新属性，并将扩展后的新行添加到新的属性集上。附加的新属性是原始酒精属性的函数输出。例如，第一个新属性是（(alch - 7) * (alch - 7）/10)，其中 alch 是数据行的酒精级别。引入常数 7 和 10，从而使新产生的属性都能在一个图上画出来。基本上，新属性取酒精值的平方。

该过程的下一步是使用扩展属性集合来构建一个线性分类器（或者其他可用方法来构建线性模型）。不论使用哪种方法来构建线性模型，模型都需要针对每个属性引入乘子（或者系数），包括新属性。如果扩展使用的函数都是原始属性的幂，线性模型的输出就是原始属性幂的系数。通过选择扩展函数，其他函数族也可以用于构建线性分类器。

图 4-8 展示了新属性与原始属性的依赖关系。可以看到扩展属性是原始属性的平方、log 以及正弦。

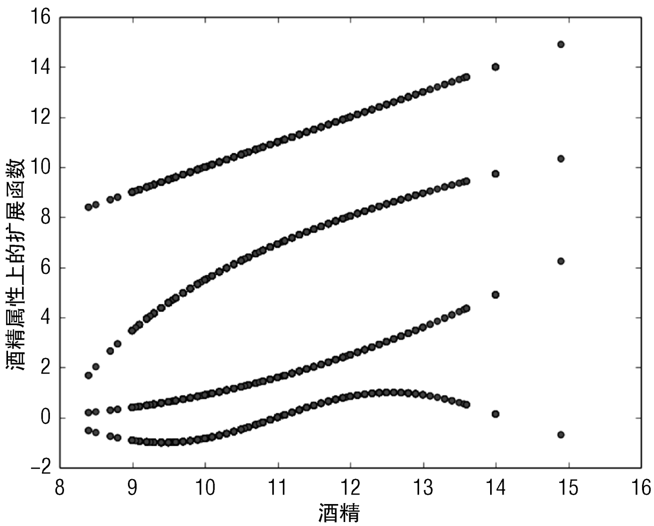


图 4-8　用于生成酒精扩展属性的函数

4.4.4 向线性方法中引入非数值属性

惩罚线性回归方法（以及其他线性方法）需要使用数值属性。如果你的问题包含其他非数值属性（也称作类别或者因子属性）该怎么办？一个熟悉的例子是性别属性，属性的可能值只有男和女。进行类别属性转换的标准方法是将属性的可能取值编码为若干新的属性列。如果一个属性有 N 个可能值，那么该属性将被编码为 N-1 列新的属性。对于每行记录，如果原始属性值为第 i 个可能值，那么将对应的新属性列的第 i 列设为 1，其他列设为 0。如果该行的原始属性值为第 N 个值，则将对应新属性的所有列设为 0。

代码清单 4-6 展示了如何在鲍鱼数据集上应用该技术。数据集对应的任务是基于不同的身体指标来预测鲍鱼年龄。

代码清单 4-6 在惩罚线性回归方法中对类别属性进行编码 -Abalone Data-larsAbalon.py

```
__author__ = 'mike_bowles'

import urllib2
from pylab import *
import matplotlib.pyplot as plot

target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/abalone/abalone.data")
#read abalone data
data = urllib2.urlopen(target_url)

xList = []
labels = []

for line in data:
    #split on semi-colon
    row = line.strip().split(",")

    #put labels in separate array and remove label from row
    labels.append(float(row.pop()))

    #form list of list of attributes (all strings)
    xList.append(row)

names = ['Sex', 'Length', 'Diameter', 'Height', 'Whole weight', \
    'Shucked weight', 'Viscera weight', 'Shell weight', 'Rings']
```

```
#code three-valued sex attribute as numeric
xCoded = []
for row in xList:
    #first code the three-valued sex variable
    codedSex = [0.0, 0.0]
    if row[0] == 'M': codedSex[0] = 1.0
    if row[0] == 'F': codedSex[1] = 1.0

    numRow = [float(row[i]) for i in range(1,len(row))]
    rowCoded = list(codedSex) + numRow
    xCoded.append(rowCoded)
namesCoded = ['Sex1', 'Sex2', 'Length', 'Diameter', 'Height', \
    'Whole weight', 'Shucked weight', 'Viscera weight', \
    'Shell weight', 'Rings']

nrows = len(xCoded)
ncols = len(xCoded[1])

xMeans = []
xSD = []
for i in range(ncols):
    col = [xCoded[j][i] for j in range(nrows)]
    mean = sum(col)/nrows
    xMeans.append(mean)
    colDiff = [(xCoded[j][i] - mean) for j in range(nrows)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrows)])
    stdDev = sqrt(sumSq/nrows)
    xSD.append(stdDev)

#use calculate mean and standard deviation to normalize xCoded
xNormalized = []
for i in range(nrows):
    rowNormalized = [(xCoded[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncols)]
    xNormalized.append(rowNormalized)

#Normalize labels
meanLabel = sum(labels)/nrows
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] -
```

```
        meanLabel) for i in range(nrows)])/nrows)

    labelNormalized = [(labels[i] - meanLabel)/sdLabel \
        for i in range(nrows)]

    #initialize a vector of coefficients beta
    beta = [0.0] * ncols

    #initialize matrix of betas at each step
    betaMat = []
    betaMat.append(list(beta))

    #number of steps to take
    nSteps = 350
    stepSize = 0.004
    nzList = []

    for i in range(nSteps):
        #calculate residuals
        residuals = [0.0] * nrows
        for j in range(nrows):
            labelsHat = sum([xNormalized[j][k] * beta[k]
                for k in range(ncols)])
            residuals[j] = labelNormalized[j] - labelsHat

        #calculate correlation between attribute columns from
        #normalized wine and residual
        corr = [0.0] * ncols

        for j in range(ncols):
            corr[j] = sum([xNormalized[k][j] * residuals[k]
                for k in range(nrows)]) / nrows

        iStar = 0
        corrStar = corr[0]

        for j in range(1, (ncols)):
            if abs(corrStar) < abs(corr[j]):
                iStar = j; corrStar = corr[j]
```

```
        beta[iStar] += stepSize * corrStar / abs(corrStar)
        betaMat.append(list(beta))


        nzBeta = [index for index in range(ncols) if beta[index] != 0.0]
        for q in nzBeta:
            if (q in nzList) == False:
                nzList.append(q)

    nameList = [namesCoded[nzList[i]] for i in range(len(nzList))]

    print(nameList)
    for i in range(ncols):
        #plot range of beta values for each attribute
        coefCurve = [betaMat[k][i] for k in range(nSteps)]
        xaxis = range(nSteps)
        plot.plot(xaxis, coefCurve)

    plot.xlabel("Steps Taken")
    plot.ylabel(("Coefficient Values"))
    plot.show()

    Printed Output - [filename- larsAbaloneOutput.txt]
    ['Shell weight', 'Height', 'Sex2', 'Shucked weight', 'Diameter', 'Sex1']
```

第一个属性是鲍鱼的性别，有 3 种可能值。因为鲍鱼出生不久，性别不能确定，所以有 3 种可能值为 M（雄）、F（雌）以及 I（未定）。

和列相关的变量名使用 Python 列表展示。对于鲍鱼数据集，这些列名并不在数据文件的第一行出现，而是单独保存为一个文件放在 UC Irvine 网站上。列表的第一个变量是动物性别。列表最后一个变量是环数，即切开鲍鱼的壳，使用显微镜观察并统计鲍鱼壳上环的个数。环数实际代表鲍鱼的年龄。该问题的目标是训练一个回归系统来预测环数，而非直接对环数进行计数，这种预测相当于一种更简单、更省时以及更经济的测量手段。

在属性矩阵进行归一化之前，需要完成对性别属性的编码。该过程是构建 2 列来代表 3 种可能的值。构建逻辑是如果对应行的性别为雄性（M），将第 1 列设为 1，否则设为 0。如果性别为雌性（F），第 2 列设为 1。如果样本还处在幼年，2 列都设为 0。新属性列的列名为 Sex1 和 Sex2。

一旦编码完成，属性矩阵包含所有数值，样本就可以按照之前的方法进行处理。算

法将变量归一化为 0 均值以及标准差，然后应用先前提到的 LARS 算法来推断系数曲线。输出展示了进入惩罚线性回归模型中的变量顺序。你会观察到用于编码性别的新的 2 列属性都出现在解中。

图 4-9 为 LARS 算法针对该问题生成的系数曲线。第 5 章针对该问题会使用多种方法来提升效果。

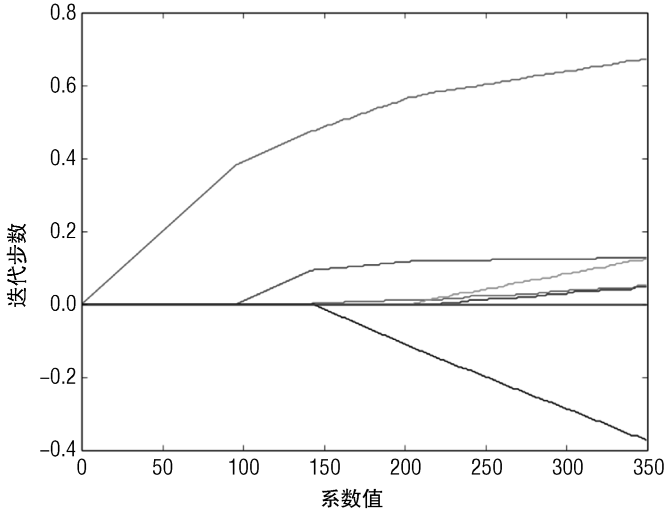


图 4-9 对类别变量使用编码后的新属性在鲍鱼数据集上应用 LARS 算法训练得到的系数曲线

本章讨论了惩罚线性回归的几种扩展方法，拓宽了其适应不同问题的能力。本节描述了一个简单常用方法，即将分类问题转换为普通线性回归问题。讨论了如何将二分类推广到多分类问题上。接着讨论了如何在原始属性上应用非线性函数生成新属性，将新属性添加到模型，从而实现使用线性回归来建模非线性行为。最后展示了如何将类别变量转换为数值变量，从而可以在类别变量上训练线性算法。类别变量的转换方法不止可以用于线性回归，也可以用于其他线性方法，如支持向量机。

小结

本章的目标是打基础，让你可以自信地使用 Python 包来实现算法。本章将输入数据描述为一个用于表示结果的列向量和一个用于表示属性的矩阵。第 3 章提到预测模型的复杂度需要与问题复杂度以及数据集规模相一致，并且给出线性回归模型的调参方法。本章在此基础上介绍了几种最小化算法，其中可调的系数惩罚项被添加到最小二乘法的错误惩罚项中。正如本章所展示的，利用系数个数作为惩罚项可以对系数进行压缩，从而

实现对模型复杂度的调整。我们看到如何使用样本外数据上的错误来调整模型的复杂度，从而获得最优性能。

本章描述了两种当代方法用于求解惩罚线性回归最小化问题，介绍如何使用 Python 来实现算法，从而帮助你掌握算法的核心代码。本章以普通回归问题（数值特征以及数值目标）作为例子对算法进行深度介绍，也介绍了线性回归的几种扩展方法，扩大了线性回归的使用场景，这些扩展包括解决二分类问题、多分类问题、属性与结果非线性关系的问题以及非数值属性问题。

第 5 章将使用 Python 包来解决一系列精心挑选的问题，从而巩固学习到的内容。通过本章所学内容，相信你已经对 Python 包中的不同参数和方法熟悉了很多。

参考文献

1. Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani（2004）. “Least Angle Regression.” A nnals of Statistics , 32（2）, 407-499.

2. Jerome H. Friedman, Trevor Hastie and Rob Tibshirani（2010）. “Regularization Paths for Generalized Linear Models via Coordinate Descent.” Journal of Statistical Software, vol. 33, issue 1, Feb 2010.

第 5 章

使用惩罚线性方法来构建预测模型

第 2 章我们了解了多个不同的数据集，目标是理解数据，理解不同属性与预测标签之间的关系，理解问题本质。本章将再次使用这些数据，通过一些例子来展示使用惩罚线性回归方法来构建预测模型的过程。一般来讲，模型构建可以分为两个或者多个阶段。

第 4 章我们提到的构建惩罚线性回归模型包含二步。第一步是在整个数据集上训练获得系数曲线。第二步是运行交叉验证来寻找最佳的样本外性能，并提取该性能对应的模型。确定模型能达到的最高性能是模型设计最难的部分，对于本章的绝大部分例子，我们只呈现第 2 步。在整个数据集上进行训练是为了得到最佳的模型系数，这并不会改变对错误（或者说算法性能）的估计。

本章将在一系列不同的问题上运行算法：回归问题、分类问题、包含类别属性的问题、以及标签与属性存在非线性关系的问题。本章也会进一步验证基扩展是否会提升预测性能。对每个例子，本章都会介绍为了达到一个可部署的线性模型，中间会采取的关键步骤，也会考虑一些备选方案，目的是得到最佳性能。

5.1 惩罚线性回归的 Python 包

第 4 章的例子使用如下的训练算法：LARS 以及使用 ElasticNet 惩罚的共轭梯度下降方法。第 4 章使用 Python 代码的目的是展示算法的工作原理，从而深入理解。幸运的是，不需要在每次使用时都重复编写算法。

Scikit-learn 算法包已经实现了套索、LARS 以及 ElasticNet 回归。使用算法包有两个优势：一是减少要编写以及调试的代码行数，另一个优势是包的运行速度要远快于第 4 章的代码。

scikit-learn 算法包会利用一些已经证明的实用方法，如去掉对无关属性关联的计算来减少计算量。你会看到这些算法包的执行速度非常快。

本章使用的算法包含在 sklearn.linear_model 算法包下，对应链接为 http://scikit-learn.org/stable/modules/classes.html#modulesklearn。linear_model 展示了可供使用的模型清单。注意到几种模型的风格略微不同。例如，linear_model.ElasticNet 包和 linear_model.

ElasticNetCV 包对应本章介绍的 2 个任务。Python 包 linear_model.ElasticNet 用于在整个数据集上计算系数曲线，linear_model.ElasticNetCV 通过交叉验证生成对 ElasticNet 模型性能的样本外估计。这些包都是现成的，可以直接使用。

两个版本的包都使用相同的输入格式（两个 numpy 数组：一个表示属性，另一个表示标签）。在一些情况下，为了更精细地控制交叉验证每一折验证使用的训练集和测试集，可能不能直接使用交叉验证的函数版本。

- 如果问题包含一个类别属性，并且该类别属性的某个属性值出现较少，就需要通过抽样来确保每份数据中包含特定属性值的样本个数是等比例的。
- 此外，也可能需要访问每折数据来计算错误统计量，比如不想使用 CV 包中的均方误差（MSE），想换用其他错误统计量，如平均绝对误差（MAE），因为它能更好地代表实际问题中的错误。
- 另一种需要对每折数据计算错误的情况是使用线性回归来解决分类问题。正如第 3 章所述，分类问题使用的标准错误指标一般是误分类率或者 ROC 曲线下面积（AUC）。可以在本章的岩石 - 水雷分类以及玻璃分类的例子中看到。

学习使用这些包尤其要注意以下几点：一是一些包（不是所有包）在模型拟合前会自动对属性进行归一化；二是 scikit-learn 包中变量的命名规则与第 4 章中以及 Friedman 论文中的变量命名规则都不同。第 4 章使用变量 λ 来代表系数惩罚项的乘子，使用变量 α 来代表套索惩罚项以及 ElasticNet 中岭回归惩罚项的比例。scikit-learn 包使用 α 来取代 λ，使用 l1_ratio 来取代 α。下面介绍 scikit-learn 包使用的记号。

SCIKIT-LEARN 的改动

scikit-learn 的目标是通过正则化等技术将所有的惩罚线性回归算法归结为统一的形式。在写作本书时，该工作仍在进行当中。

5.2 多变量回归：预测红酒口感

正如第 2 章所讨论的，红酒口感数据集来源于 UC Irvine 数据仓库（http://archive.ics.uci.edu/ml/datasets/Wine+Quality）数据集包含 1599 种红酒的化学分析以及每种红酒的平均口感得分（通过多人品尝）。预测问题是给定化学成分来预测口感。化学成分数据包含 11 种不同的化学属性值：酒精内容、pH 以及柠檬酸等，可以到第 2 章或者 UC Irvine 的网站去了解该数据集的更多细节。

预测红酒口感是一个回归问题，因为问题目标是预测质量得分，质量得分为 0 ～ 10 的整数。数据集只包括得分为 3 ～ 8 的样例。因为只给出了整数得分，所以理论上

也可以将该问题转换为多分类问题，即该问题包含 6 种可能的类别（整数 3 ～ 8）。但是作为分类问题会忽略不同得分之间存在的顺序关系（如 5 比 6 差，但比 4 好），所以回归是一个更自然的解决方法，因为它的预测很好地保留了顺序关系。

定义问题的另一种方式是从错误指标出发，这些指标关联到回归问题或者是多分类问题。回归的错误函数是均方误差。当实际的口感得分为 3 时，预测值为 5 要比预测值为 4 对累积错误的贡献更大。多分类问题的错误指标是被误分的样本数目。使用该错误指标，如果实际的口感为 3，预测 5 或者 4 对于预测错误的贡献相同。回归显得更加自然，但是没有更好的办法证明它的性能更优。唯一能确定哪种方法效果更好的方式是两种都尝试一下。在“多类别分类：分类犯罪场景下的玻璃样本”，会学习如何处理多类问题。接着可以再回来尝试多分类方法，看看使用分类，效果会变好还是变差。实际你会用哪种错误指标?

5.2.1 构建并测试模型以预测红酒口感

构建模型的第一步是通过样本外的性能来判断模型能否满足性能要求。代码清单 5-1 展示了执行 10 折交叉验证的效果并绘制了图形。代码的第 1 节从 UCI 网站读入数据，转化为列表的列表，然后对属性以及标签进行归一化。接着将列表转换为 numpy 数组 X（属性矩阵）以及数组 Y（标签向量）。回归存在 2 个版本的定义。其中一个版本使用归一化的数值，另一个版本使用非归一化的数值。不论哪种定义，都可以将对应的非归一化版本注释掉，重新运行代码看对属性或者标签进行归一化的实际效果。使用一行代码定义交叉验证的数据份数（10），并且对模型进行了训练。然后程序在 10 份数据的每一份上绘制错误随 α 变化的曲线，同时绘制在 10 份数据上的错误平均值。图 5-1 ～图 5-3 为使用不同归一化版本的预测效果，这 3 个例子对应的归一化情况如下。

（1）X 做归一，Y 不做归一。

（2）X 和 Y 都做归一。

（3）X 和 Y 都不做归一。

代码清单 5-1 在红酒口感数据集上，使用交叉验证来估计套索模型的样本外错误

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import datasets, linear_model
from sklearn.linear_model import LassoCV
from math import sqrt
import matplotlib.pyplot as plot
```

```
#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

#Normalize columns in x and labels
#Note: be careful about normalization. Some penalized
#regression packages include it and some don't.
nrows = len(xList)
ncols = len(xList[0])

#calculate means and variances
xMeans = []
xSD = []
for i in range(ncols):
    col = [xList[j][i] for j in range(nrows)]
    mean = sum(col)/nrows
    xMeans.append(mean)
    colDiff = [(xList[j][i] - mean) for j in range(nrows)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrows)])
    stdDev = sqrt(sumSq/nrows)
    xSD.append(stdDev)
```

```
#use calculate mean and standard deviation to normalize xList
xNormalized = []
for i in range(nrows):
    rowNormalized = [(xList[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncols)]
    xNormalized.append(rowNormalized)

#Normalize labels
meanLabel = sum(labels)/nrows

sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] -
        meanLabel) for i in range(nrows)])/nrows)

labelNormalized = [(labels[i] - meanLabel)/sdLabel \
        for i in range(nrows)]

#Convert list of list to np array for input to sklearn packages

#Unnormalized labels
Y = numpy.array(labels)

#normalized lables
Y = numpy.array(labelNormalized)

#Unnormalized X's
X = numpy.array(xList)

#Normalized Xss
X = numpy.array(xNormalized)

#Call LassoCV from sklearn.linear_model
wineModel = LassoCV(cv=10).fit(X, Y)

# Display results

plot.figure()
plot.plot(wineModel.alphas_, wineModel.mse_path_, ':')
plot.plot(wineModel.alphas_, wineModel.mse_path_.mean(axis=-1),
         label='Average MSE Across Folds', linewidth=2)
plot.axvline(wineModel.alpha_, linestyle='--',
```

```
        label='CV Estimate of Best alpha')
plot.semilogx()
plot.legend()
ax = plot.gca()
ax.invert_xaxis()
plot.xlabel('alpha')
plot.ylabel('Mean Square Error')
plot.axis('tight')
plot.show()

#print out the value of alpha that minimizes the Cv-error
print("alpha Value that Minimizes CV Error ",wineModel.alpha_)
print("Minimum MSE ", min(wineModel.mse_path_.mean(axis=-1)))

Printed Output: Normalized X, Un-normalized Y
('alpha Value that Minimizes CV Error ', 0.010948337166040082)
('Minimum MSE ', 0.433801987153697)

Printed Output: Normalized X and Y
('alpha Value that Minimizes CV Error ', 0.013561387700964642)
('Minimum MSE ', 0.66558492060028562)

Printed Output: Un-normalized X and Y
('alpha Value that Minimizes CV Error ', 0.0052692947038249062)
('Minimum MSE ', 0.43936035436777832)
```

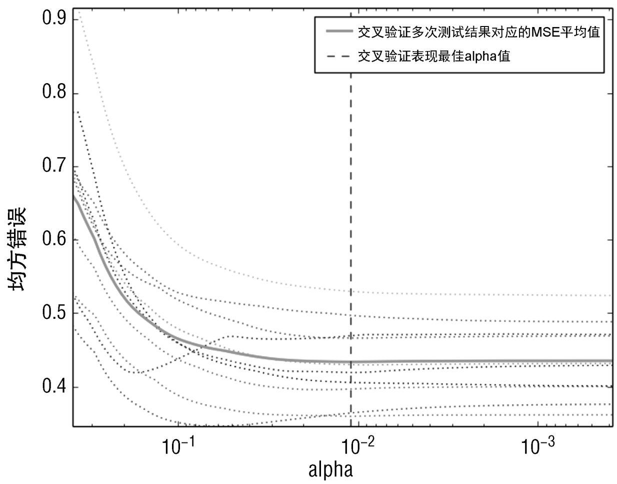


图 5-1 在红酒口感数据集上应用套索模型：基于非归一化的 *Y* 得到的样本外错误

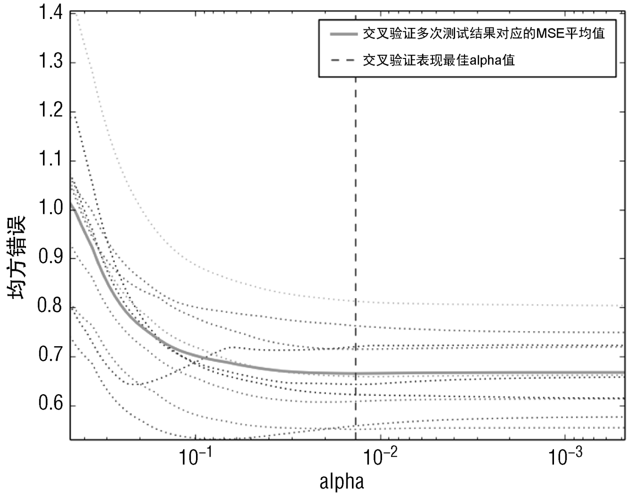


图 5-2　在红酒口感数据集上应用套索模型：基于归一化的 *Y* 得到的样本外错误

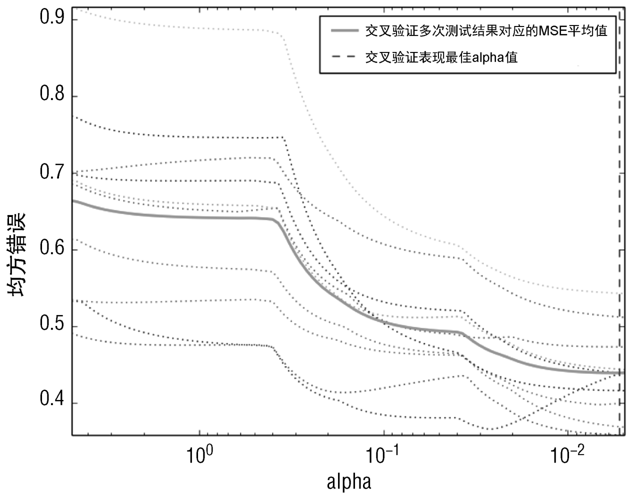


图 5-3　在红酒口感数据集上应用套索模型：基于归一化的 *X* 和 *Y* 得到的样本外错误

代码清单 5-1 最下边的输出展示了使用归一化 *Y* 带来的 MSE 的显著增长。相比之下，图 5-1 以及图 5-2 形状类似。它们之间的唯一差别是 *Y* 轴的数据尺度。参照代码清单 2-13 可以看到非归一化的红酒质量得分的标准偏差为 0.81。这意味着归一化到标准偏差 1.0 需要大概乘以 1.2。这会导致 MSE 增长 1.2^2 倍。对标签进行归一化会使 MSE 失去与原始数据的关联。抽取 MSE 平方根并转换回标签原始尺度，这些实现都是现成的。在本例中，MSE（基于归一化的 *Y*）为 0.433。平方根约为 0.64。

这意味着 +/-sigma 的错误大约在 1.3 倍个单位口感得分范围内。所以，Y 做归一化并不会对结果有本质差异。X 做归一化的效果如何？ X 做归一化是会提高，还是降低性能呢？

代码清单 5-1 中的数字集合展示了 X 不做归一化导致 MSE 轻微增长。然而在图 5-3 中，CV 错误随 alpha 的变化关系与图 5-1 和图 5-2 相比存在显著差异。图上的扇形变化是由 X 特征尺度混乱（未做归一化）导致，算法挑选了尺度大的变量进行预测，对应的系数很小。这种情况发生的条件是如果变量与 Y 的关联很大，或者变量与 Y 关联很小，但是特征尺度很大。算法使用一个较差的变量进行若干次迭代，直到 α（之前用 λ 代替）变得足够小以引入一个更好的变量，这时错误才会陡然下降。这个例子的教训就是需要对 X 做归一化，X 不做归一化一定要足够警惕。

5.2.2 部署前在整个数据集上进行训练

代码清单 5-2 为在整个数据集上进行训练的代码。正如之前提到的，在整个数据集上进行训练是为了获得部署使用的最佳权重系数。交叉验证可以对模型性能进行评估，同时获得性能最好的 α 参数值。将数据从 UC Irvine 数据仓库读入并且归一化后，程序将数据转换为 numpy array 类型，然后调用 lasso_path 方法来生成 α 值（即惩罚项权重）以及对应的特征系数。系数的变化轨迹如图 5-4 所示。

代码清单 5-2 在完整数据集上训练套索模型 -wineLassoCoefCurves.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import datasets, linear_model
from sklearn.linear_model import LassoCV
from math import sqrt
import matplotlib.pyplot as plot

#read data into iterable
target_url = "http://archive.ics.uci.edu/ml/machine-learning-databases/
wine-quality/winequality-red.csv"
data = urllib2.urlopen(target_url)

xList = []
labels = []
```

```
names = []
firstLine = True
for line in data:
    if firstLine:
       names = line.strip().split(";")
       firstLine = False
    else:
       #split on semi-colon
       row = line.strip().split(";")
       #put labels in separate array
       labels.append(float(row[-1]))
       #remove label from row
       row.pop()
       #convert row to floats
       floatRow = [float(num) for num in row]
       xList.append(floatRow)

#Normalize columns in x and labels
#Note: be careful about normalization. Some penalized regression
#packages include it and some don't.

nrows = len(xList)
ncols = len(xList[0])

#calculate means and variances
xMeans = []
xSD = []
for i in range(ncols):
    col = [xList[j][i] for j in range(nrows)]
    mean = sum(col)/nrows
    xMeans.append(mean)
    colDiff = [(xList[j][i] - mean) for j in range(nrows)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrows)])
    stdDev = sqrt(sumSq/nrows)
    xSD.append(stdDev)

#use calculate mean and standard deviation to normalize xList
xNormalized = []
```

```
for i in range(nrows):
    rowNormalized = [(xList[i][j] - xMeans[j])/xSD[j] for j in
    range(ncols)]
    xNormalized.append(rowNormalized)

#Normalize labels
meanLabel = sum(labels)/nrows
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] - meanLabel)
for i in range(nrows)])/nrows)

labelNormalized = [(labels[i] - meanLabel)/sdLabel for i in
range(nrows)]

#Convert list of list to np array for input to sklearn packages

#Unnormalized labels
Y = numpy.array(labels)

#normalized lables
Y = numpy.array(labelNormalized)

#Unnormalized X's
X = numpy.array(xList)

#Normalized Xss
X = numpy.array(xNormalized)

alphas, coefs, _ = linear_model.lasso_path(X, Y, return_models=False)


plot.plot(alphas,coefs.T)

plot.xlabel('alpha')
plot.ylabel('Coefficients')
plot.axis('tight')
plot.semilogx()
ax = plot.gca()
```

```
ax.invert_xaxis()
plot.show()

nattr, nalpha = coefs.shape
#find coefficient ordering
nzList = []
for iAlpha in range(1,nalpha):
    coefList = list(coefs[: ,iAlpha])
    nzCoef = [index for index in range(nattr) if coefList[index] != 0.0]
    for q in nzCoef:
        if not(q in nzList):
            nzList.append(q)

nameList = [names[nzList[i]] for i in range(len(nzList))]
print("Attributes Ordered by How Early They Enter the Model", nameList)

#find coefficients corresponding to best alpha value. alpha value
# corresponding to normalized X and normalized Y is 0.013561387700964642

alphaStar = 0.013561387700964642
indexLTalphaStar = [index for index in range(100) if alphas[index] >
alphaStar]
indexStar = max(indexLTalphaStar)

#here's the set of coefficients to deploy
coefStar = list(coefs[:,indexStar])
print("Best Coefficient Values ", coefStar)

#The coefficients on normalized attributes give another slightly
#different ordering

absCoef = [abs(a) for a in coefStar]

#sort by magnitude
coefSorted = sorted(absCoef, reverse=True)

idxCoefSize = [absCoef.index(a) for a in coefSorted if not(a == 0.0)]
```

```
namesList2 = [names[idxCoefSize[i]] for i in range(len(idxCoefSize))]

print("Attributes Ordered by Coef Size at Optimum alpha", namesList2)


Printed Output w. Normalized X:
('Attributes Ordered by How Early They Enter the Model',
['"alcohol"', '"volatile acidity"', '"sulphates"',
'"total sulfur dioxide"', '"chlorides"', '"fixed acidity"', '"pH"',
'"free sulfur dioxide"', '"residual sugar"', '"citric acid"',
'"density"'])

('Best Coefficient Values ',
[0.0, -0.22773815784738916, -0.0, 0.0, -0.094239023363375404,
0.022151948563542922, -0.099036391332770576, -0.0,
-0.067873612822590218, 0.16804102141830754, 0.37509573430881538])

('Attributes Ordered by Coef Size at Optimum alpha',
['"alcohol"', '"volatile acidity"', '"sulphates"',
'"total sulfur dioxide"', '"chlorides"', '"pH"',
'"free sulfur dioxide"'])

Printed Output w. Un-normalized X:
('Attributes Ordered by How Early They Enter the Model',
['"total sulfur dioxide"', '"free sulfur dioxide"', '"alcohol"',
'"fixed acidity"', '"volatile acidity"', '"sulphates"'])

('Best Coefficient Values ', [0.044339055570034182, -1.0154179864549988,
0.0, 0.0, -0.0, 0.0064112885435006822, -0.0038622920281433199, -0.0,
-0.0, 0.41982634135945091, 0.37812720947996975])

('Attributes Ordered by Coef Size at Optimum alpha',
['"volatile acidity"', '"sulphates"', '"alcohol"', '"fixed acidity"',
'"free sulfur dioxide"', '"total sulfur dioxide"'])
```

上述程序对 α 值进行硬编码，使用交叉验证中效果最好的 α 值。代码使用的版本是基

于归一化属性以及标签的最佳 alpha 值。如果属性取非归一化的值，对应的最佳的 α 值也会改变。Y 不做归一化，α 值会乘以 1.2，因为归一化后的 Y 的偏差为 1.0（参照之前讨论的标签归一化以及不归一化对 MSE 差异的影响）。使用硬编码的 α 值来计算对应于最佳交叉验证结果的系数向量。

代码清单 5-2 展示了 3 种设置的输出：基于归一化属性以及非归一化的标签、全部归一化、全部非归一化。每种设置的打印输出包含一列属性，这些属性是随 α 降低，逐步挑选出来加入模型中的属性（Python 包中的 α 值对应于第 4 章中的惩罚项 λ）。打印输出展示了对应于硬编码的 α 值的特征系数。打印输出的第三个元素是根据系数大小得到的属性排序（使用指定的 α 值）。属性系数取值是另一种反映属性重要性的方式，这种排序只有当属性归一化后才有意义。注意到使用归一化属性，前面讨论的为属性重要性赋值的 2 种方法生成的属性顺序基本一致，在不太重要的属性上可能存在一些差异。如果使用非归一化的属性，则得到的结果离正确值差之甚远。

正如前面提到的，变量进入解的顺序（随着 α 减少）受属性是否归一化影响较大。如果一个变量没有被归一化，属性的数值量级而不是属性的内在价值会决定属性的使用情况。通过比较基于归一化属性的变量顺序与非归一化属性的变量顺序可以很明显地发现这个问题。

图 5-4 和图 5-5 分别为基于归一化属性以及非归一化属性，套索模型的系数曲线。基于非归一化属性的系数曲线相较于归一化属性的系数曲线更加无序。几个早期进入解的系数相对于后续进入解的系数更接近于 0。这种现象正好证明了系数进入模型的顺序与最佳解的系数尺度的顺序存在本质不同。

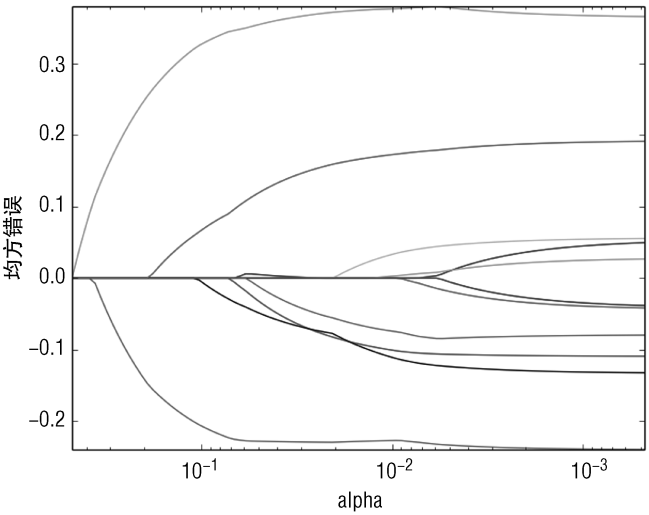


图 5-4 用于预测红酒品质的套索模型的系数曲线

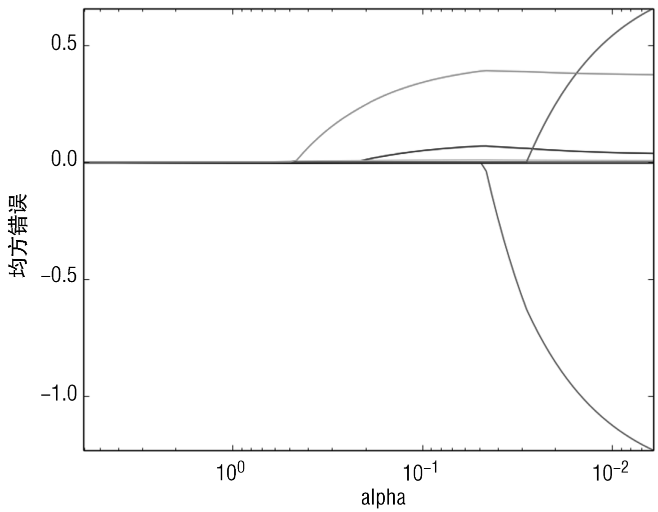


图 5-5 在非归一化的 *X* 上训练套索模型得到的系数曲线

5.2.3 基扩展：基于原始属性扩展新属性来改进性能

第 4 章讨论了在原始属性通过函数来扩展新属性。这么做的目的是提高性能。代码清单 5-3 展示了如何添加 2 个新属性到红酒数据上。

代码清单 5-3 使用样本外错误来评估新属性对红酒品质的预测效果 -wineExpaned LassoCV.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import datasets, linear_model
from sklearn.linear_model import LassoCV
from math import sqrt
import matplotlib.pyplot as plot

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
```

```
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

#append square of last term (alcohol)

for i in range(len(xList)):
    alcElt = xList[i][-1]
    volAcid = xList[i][1]
    temp = list(xList[i])
    temp.append(alcElt*alcElt)
    temp.append(alcElt*volAcid)
    xList[i] = list(temp)

#add new name to variable list
names[-1] = "alco^2"
names.append("alco*volAcid")

#Normalize columns in x and labels
#Note: be careful about normalization. Some penalized regression
packages include it and some don't.

nrows = len(xList)
ncols = len(xList[0])

#calculate means and variances
```

```
xMeans = []
xSD = []
for i in range(ncols):
    col = [xList[j][i] for j in range(nrows)]
    mean = sum(col)/nrows
    xMeans.append(mean)
    colDiff = [(xList[j][i] - mean) for j in range(nrows)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrows)])
    stdDev = sqrt(sumSq/nrows)
    xSD.append(stdDev)

#use calculate mean and standard deviation to normalize xList
xNormalized = []
for i in range(nrows):
    rowNormalized = [(xList[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncols)]
    xNormalized.append(rowNormalized)

#Normalize labels
meanLabel = sum(labels)/nrows
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] - meanLabel) \
        for i in range(nrows)])/nrows)

labelNormalized = [(labels[i] - meanLabel)/sdLabel \
        for i in range(nrows)]

#Convert list of list to np array for input to sklearn packages

#Unnormalized labels
Y = numpy.array(labels)

#normalized labels
#Y = numpy.array(labelNormalized)

#Unnormalized X's
X = numpy.array(xList)

#Normalized Xss
X = numpy.array(xNormalized)
```

```
#Call LassoCV from sklearn.linear_model
wineModel = LassoCV(cv=10).fit(X, Y)

# Display results

plot.figure()
plot.plot(wineModel.alphas_, wineModel.mse_path_, ':')
plot.plot(wineModel.alphas_, wineModel.mse_path_.mean(axis=-1),
         label='Average MSE Across Folds', linewidth=2)
plot.axvline(wineModel.alpha_, linestyle='--',
            label='CV Estimate of Best alpha')
plot.semilogx()
plot.legend()
ax = plot.gca()
ax.invert_xaxis()
plot.xlabel('alpha')
plot.ylabel('Mean Square Error')
plot.axis('tight')
plot.show()

#print out the value of alpha that minimizes the CV-error
print("alpha Value that Minimizes CV Error ",wineModel.alpha_)
print("Minimum MSE ", min(wineModel.mse_path_.mean(axis=-1)))

Printed Output: [filename - wineLassoExpandedCVPrintedOutput.txt]
('alpha Value that Minimizes CV Error ', 0.016640498998569835)
('Minimum MSE ', 0.43452874043020256)
```

属性读入后的关键步骤是将属性值转换为浮点数值。前面的很多行代码都是用于读入属性行，将其中的酒精以及挥发酸属性值读出来，然后添加酒精的平方以及酒精乘以挥发酸这两个新属性，这么做是因为这些属性对于解非常重要。为了进一步提高性能有可能需要对重要变量进行多次组合，进行多次尝试。

结果显示添加这些新变量会轻微降低性能。有时稍作尝试就可能发现部分变量会导致完全不同的结果。对于本例，可以通过系数曲线来观察新的变量能否取代最优值对应的原始变量。这些信息可以帮助移除原始变量，使用新创建的变量。

图 5-6 为基于扩展属性集生成的交叉验证的错误曲线。新的交叉验证曲线与基于原始特征的交叉验证曲线并没有本质区别。

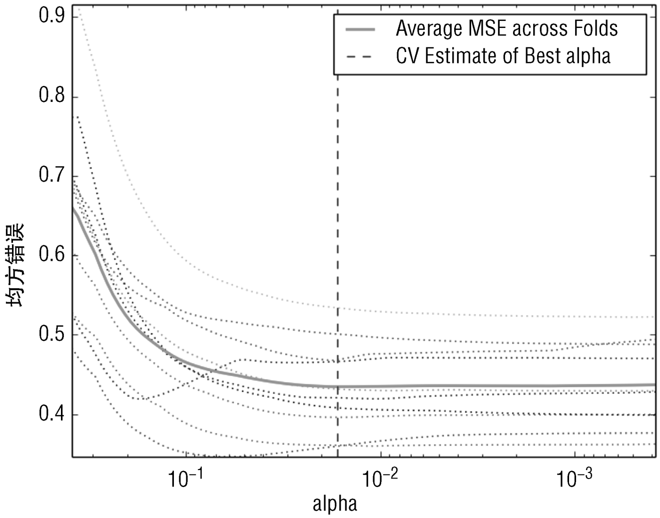


图 5-6 在红酒品质数据集上使用扩展特征集合训练套索模型得到的交叉验证错误曲线

本节介绍了在包含实数输出的问题上（回归问题）应用惩罚回归方法的过程。下一节将介绍输出为二值情况下的惩罚线性回归方法。代码类似于本节所看到的，其中一些技术（如基扩展）可以用在分类问题上，主要差别在于对性能的度量计算不同。

5.3 二分类：使用惩罚线性回归来检测未爆炸的水雷

第 4 章讨论了如何使用惩罚线性回归方法来解决分类问题，并且给出了岩石 - 水雷问题的求解过程。本节将详细介绍如何使用惩罚线性回归来求解二分类问题，具体使用 Python 的 ElasticNet 包。第 4 章讲到 ElasticNet 引入了一个更普通的惩罚函数，套索以及岭回归的惩罚函数只是该函数的一个特殊情形。随着更换惩罚函数，可以看到分类器的性能的变化。以下是求解的步骤。

（1）将二分类问题转换为回归问题。构建一个包含实数标签的向量，将其中一个类别输出设为 0，另一个类别输出设为 1。

（2）执行交叉验证。因为需要对每一份数据计算错误，交叉验证稍微复杂。Scikit-learn 包含一些便捷的功能来将这些计算流水化。

第一步（正如第 4 章所述）是将二分类问题转换为回归问题：通过将类别标签转换为实数值。岩石 - 水雷问题的目标是构建一个系统，该系统使用声纳来检测海床上的水雷。回忆第 2 章对该数据集的介绍，该数据集包含从岩石以及形状类似水雷的金属柱返回的数字信号。目标是构建一个预测系统，该系统可以对数字信号进行处理来正确识别对象是岩石还是水雷。数据集包含 208 次实验，其中 111 次实验对应的结果是水雷，97 次实验

对应的结果是岩石。数据集包含 61 列。前面 60 列是数字声纳，最后 1 列是岩石或者水雷，对应取值为 M 或者 R。前面 60 列是用于预测的属性。最后 1 列标签需要使用数值来表示（即使回归也需要数值表示）。第 4 章通过将数字 1 赋给其中一个标签、将 0 赋给另一个标签来构建数值标签。代码清单 5-4 初始化了一个空的标签列表，将每个 M 行的标签值设为 1，将每个 R 行的标签值设为 0。

有了数值属性以及数值标签，就可以使用回归版本的惩罚线性回归方法了。下一步是执行交叉验证，以获得对模型在样本外数据上的性能估计，并找到最佳的惩罚项参数 α。对于这个问题，交叉验证需要构建一个交叉验证循环来封装训练集以及测试集。为什么要构建一个交叉验证循环，而不是使用 Python 中现成的交叉验证包（类似于本章前面提到的红酒品质预测的例子）？

面向回归的交叉验证基于 MSE。MSE 对于回归问题是合理的，但对于分类问题则不是。正如第 3 章所讨论的，分类问题与回归问题使用的性能度量指标不同。第 3 章讨论了度量性能几种方法。一种自然的度量方式是计算误分类样本所占的比例。另一种方式是度量 AUC。参照第 3 章或者 Wikipedia 页面 http://en.wikipedia.org/wiki/Receiver_operating_characteristic 来回顾 AUC 指标的计算方法。为了计算指标，需要访问交叉验证中的每一份数据，并获取其中的所有预测值以及实际值。不能只基于一份数据的 MSE 来评估误分类错误。

交叉验证循环将数据切分为训练集以及测试集，然后调用 Python 的 enet_path 方法在训练数据上完成训练。程序的两个输入与默认输入不同。一个是 l1_ratio，该值被设置为 0.8。该参数决定了系数绝对值和的惩罚项占所有惩罚项的权重比例。0.8 代表惩罚项使用 80% 的绝对值和以及 20% 的均方误差和。另一个需要指定的参数是 fit_intercept，该值被设置为 False。代码使用归一化的标签以及归一化的属性。因为所有这些属性都是 0 均值的，所以不需要计算插值项。只有属性以及标签期望存在偏差时，才需要增加一个常数插值项。使用归一化标签来消除插值项会使预测计算更加清晰，但也会使对应 MSE 的意义不直观。不过对于分类问题，我们不会使用 MSE 来度量性能。

对每一折验证，训练得到的系数用于在测试数据上生成预测，对应代码实现使用 numpy 点乘函数、使用样本外数据的属性值以及当前训练得到的系数。2 个 numpy 数组相乘得到另外一个二维数组，该数组的行对应样本外的测试数据，该数组的列对应由 enet_path（对应于系数向量序列以及 α 序列）生成的模型序列。每一折交叉验证对应的预测矩阵被拼接到一起（视觉上相当于把一个矩阵叠加到另一个矩阵上）作为样本外数据的标签。在运行的最后，这些按折生成的样本外数据的预测结果可以被高效处理，用于生成每个模型的性能指标，从中可以挑选一个合适的复杂度（α）对应的模型进行部署。

代码清单 5-4 给出了基于两种指标的对比结果。第一个指标是误分类错误。第二个指

标是信号接收曲线的曲线下的面积（即 ROC）。预测矩阵的每一列代表使用一组模型系数在样本外数据上运算得到的预测结果。预测数据表示为 1 列是因为每次交叉验证会预留出一行数据。误分类计算使用生成的预测类别以及对应的实际标签类别（代码中称作 yOut）比较，其中生成的预测类别是通过比较预测值和固定阈值（本例中的 0.0）得到的。通过比较预测类别和实际标签 yOut，可以确定预测是否正确。图 5-7 展示了性能取到相同最小值的几个点。在性能随 α 变化的图上，选择离左边较远的点往往是一个不错方案。因为右侧的点更容易充分拟合数据。选择离左边较远的解是相对保守的方案，在这种情况下，很有可能部署错误与交叉验证的错误程度一致。

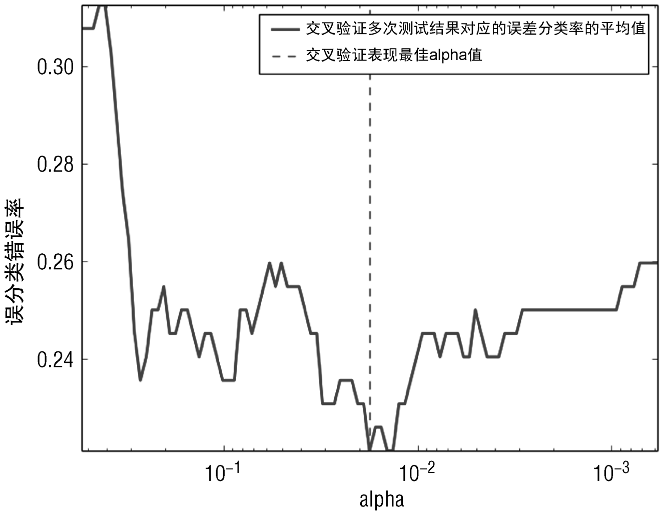


图 5-7 分类器在样本外数据上的误分类性能

度量分类器性能的另一种方式是 AUC。AUC 的优势是通过最大化 AUC，能获得最佳性能并且与应用场景无关，不论是为不同错误类型设置权重，还是重点关注某类数据的分类正确率。严格来讲，最大化 AUC 不能确保在一个特定的错误率上获得最优性能。通过对比 AUC 选择的模型、通过最小化错误率得到的模型以及观察曲线形状会帮助获得对解的信心，同时能了解到通过彻底的优化性能指标，还能在多大程度上提高性能。

代码清单 5-4 中 AUC 的计算使用来自 sklearn 的 roc_curve 以及 roc_auc_score 程序。生成 AUC 随 α 变化的曲线类似于计算误分类错误的过程，不同在于模型生成的预测值以及真正的标签值被传送给 roc_auc_score 程序来计算 AUC。图 5-8 绘制了这些 AUC 的值。生成的曲线从上往下看类似于误分类错误曲线，从上到下是因为 AUC 的值越大越好，误分类错误越小越好。代码清单 5-4 的打印输出显示基于误分类错误的最优模型与基于 AUC 的最优模型不完全相同，但它们距离不大。图 5-9 为最大化 AUC 分类器的

ROC 曲线。

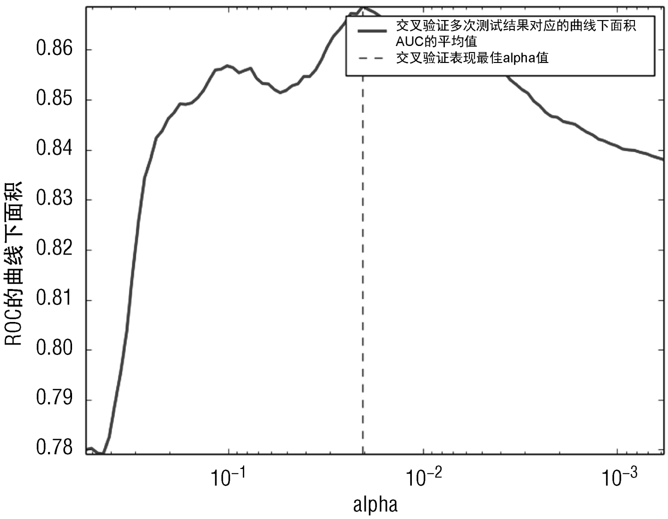


图 5-8　分类器在样本外数据上的 AUC 性能

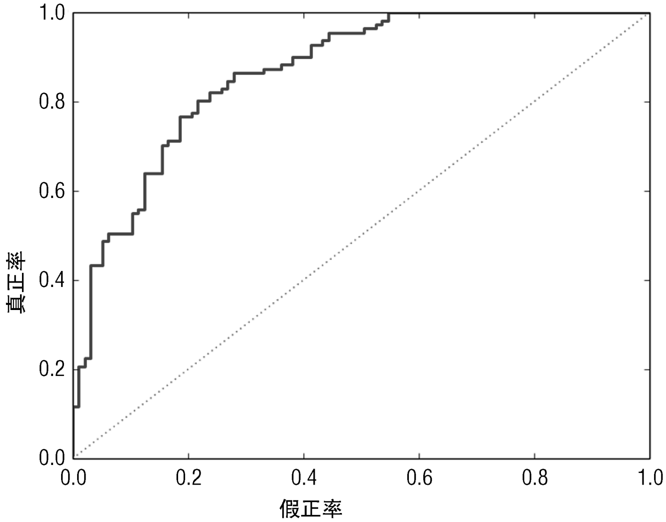


图 5-9　最佳性能分类器的 ROC 曲线

在这个问题中，一些错误相对于其他错误的可能代价更大，你想使预测结果离代价大的错误更远。对于岩石水雷问题，如果将未爆炸的水雷作为岩石，可能比将岩石误分类为水雷代价更大。

处理这类问题的一种系统性的方法是使用混淆矩阵（如第 3 章讨论的）。基于 roc_

curve 程序输出很容易构建混淆矩阵。ROC 曲线上的点对应于不同的阈值。点（1,1）对应一个极端情况，即阈值设置为最低值，所有点都被分类为水雷。这使得真正率和假负率等于 1。分类器使所有的正例在右边，但同时也错分了所有负例。将阈值设置为比所有点都高，生成了图形的另一角。为了获得点在困惑矩阵不同方框内的移动细节需要挑一些阈值然后打印结果。代码清单 5-4 展示了 3 个阈值，这些阈值对应所有阈值的分位点阈值(不包含最后点)。设置高阈值导致低的假正率和高的假负率。设置低阈值产生相反效果。将阈值设为中间值更接近于对两类错误进行平衡。

可以为每类错误赋权重、最小化总代价来获得最佳阈值。输出的 3 个混淆矩阵可以作为一个例子来说明背后的工作原理。如果假正样本和假负样本都会花费 1 美元,中间表(对应于阈值 -0.0455）给出的总代价为 46 美元，更高阈值对应的代价为 68 美元，更低阈值代价为 54 美元。然而，如果假正的代价为 10 美元，假负的代价为 1 美元，更高阈值对应代价为 113 美元，中间阈值对应的代价为 226 美元，较低阈值对应的代价为 504 美元。此时，你会希望在更细粒度上测试阈值。总体上，为了达到好的效果，需要合理设置代价，同时要确保训练集的正例与负例的比例与实际情况相一致。岩石水雷问题的样本数据来源于实验环境，可能并不完全代表港口中实际的岩石水雷数字。这个很容易通过对其中一个类别进行抽样来修复，即复制一个类别的部分样本使该类样本数占总体样本数的比例与实际部署环境中的比例相一致。

对于岩石水雷数据，对应的训练集是均衡的，即正例和负例样本数几乎相同。在一些数据集中，可能其中一个类别的样本数要更多。例如，因特网广告的点击率只占所有展示广告很小的比例，远小于 1%。通过增强样本数较少的类别，使二类比例相接近可以获得更好的训练结果。可以通过复制样本数较少的类别样本或者移除样本数较多的类别样本来达到上述目的。

代码清单 5-4 使用 ElasticNet 回归构建二分类器 -rocksVMinesENetRegCV.py

```
__author__ = 'mike_bowles'
import urllib2
from math import sqrt, fabs, exp
import matplotlib.pyplot as plot
from sklearn.linear_model import enet_path
from sklearn.metrics import roc_auc_score, roc_curve
import numpy

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
```

```
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)
#arrange data into list for labels and list of lists for attributes
xList = []


for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

#separate labels from attributes, convert from attributes from string
#to numeric and convert "M" to 1 and "R" to 0

xNum = []
labels = []

for row in xList:
    lastCol = row.pop()
    if lastCol == "M":
        labels.append(1.0)
    else:
        labels.append(0.0)
    attrRow = [float(elt) for elt in row]
    xNum.append(attrRow)

#number of rows and columns in x matrix
nrow = len(xNum)
ncol = len(xNum[1])

alpha = 1.0

#calculate means and variances
xMeans = []
xSD = []
for i in range(ncol):
    col = [xNum[j][i] for j in range(nrow)]
    mean = sum(col)/nrow
    xMeans.append(mean)
```

```
    colDiff = [(xNum[j][i] - mean) for j in range(nrow)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrow)])
    stdDev = sqrt(sumSq/nrow)
    xSD.append(stdDev)

#use calculate mean and standard deviation to normalize xNum
xNormalized = []
for i in range(nrow):
    rowNormalized = [(xNum[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncol)]
    xNormalized.append(rowNormalized)

#normalize labels to center
#Normalize labels
meanLabel = sum(labels)/nrow
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] - meanLabel) \
        for i in range(nrow)])/nrow)

labelNormalized = [(labels[i] - meanLabel)/sdLabel for i in range(nrow)]


#number of cross-validation folds
nxval = 10

for ixval in range(nxval):
    #Define test and training index sets
    idxTest = [a for a in range(nrow) if a%nxval == ixval%nxval]
    idxTrain = [a for a in range(nrow) if a%nxval != ixval%nxval]

    #Define test and training attribute and label sets
    xTrain = numpy.array([xNormalized[r] for r in idxTrain])
    xTest = numpy.array([xNormalized[r] for r in idxTest])
    labelTrain = numpy.array([labelNormalized[r] for r in idxTrain])
    labelTest = numpy.array([labelNormalized[r] for r in idxTest])
    alphas, coefs, _ = enet_path(xTrain, labelTrain,l1_ratio=0.8,
        fit_intercept=False, return_models=False)

#apply coefs to test data to produce predictions and accumulate
if ixval == 0:
```

```
        pred = numpy.dot(xTest, coefs)
        yOut = labelTest
    else:
        #accumulate predictions
        yTemp = numpy.array(yOut)
        yOut = numpy.concatenate((yTemp, labelTest), axis=0)

        #accumulate predictions
        predTemp = numpy.array(pred)
        pred = numpy.concatenate((predTemp, numpy.dot(xTest, coefs)),
            axis = 0)


#calculate misclassification error
misClassRate = []
_,nPred = pred.shape
for iPred in range(1, nPred):
    predList = list(pred[:, iPred])
    errCnt = 0.0
    for irow in range(nrow):
        if (predList[irow] < 0.0) and (yOut[irow] >= 0.0):
            errCnt += 1.0
        elif (predList[irow] >= 0.0) and (yOut[irow] < 0.0):
            errCnt += 1.0
    misClassRate.append(errCnt/nrow)
#find minimum point for plot and for print
minError = min(misClassRate)
idxMin = misClassRate.index(minError)
plotAlphas = list(alphas[1:len(alphas)])

plot.figure()
plot.plot(plotAlphas, misClassRate,
    label='Misclassification Error Across Folds', linewidth=2)
plot.axvline(plotAlphas[idxMin], linestyle='--',
            label='CV Estimate of Best alpha')
plot.legend()
plot.semilogx()
ax = plot.gca()
ax.invert_xaxis()
```

```
plot.xlabel('alpha')
plot.ylabel('Misclassification Error')
plot.axis('tight')
plot.show()



#calculate AUC.
idxPos = [i for i in range(nrow) if yOut[i] > 0.0]
yOutBin = [0] * nrow
for i in idxPos: yOutBin[i] = 1

auc = []
for iPred in range(1, nPred):
    predList = list(pred[:, iPred])
    aucCalc = roc_auc_score(yOutBin, predList)
    auc.append(aucCalc)

maxAUC = max(auc)
idxMax = auc.index(maxAUC)

plot.figure()
plot.plot(plotAlphas, auc, label='AUC Across Folds', linewidth=2)
plot.axvline(plotAlphas[idxMax], linestyle='--',
            label='CV Estimate of Best alpha')
plot.legend()
plot.semilogx()
ax = plot.gca()
ax.invert_xaxis()
plot.xlabel('alpha')
plot.ylabel('Area Under the ROC Curve')
plot.axis('tight')
plot.show()


#plot best version of ROC curve
fpr, tpr, thresh = roc_curve(yOutBin, list(pred[:, idxMax]))

ctClass = [i*0.01 for i in range(101)]
```

```
plot.plot(fpr, tpr, linewidth=2)
plot.plot(ctClass, ctClass, linestyle=':')
plot.xlabel('False Positive Rate')
plot.ylabel('True Positive Rate')
plot.show()

print('Best Value of Misclassification Error = ', misClassRate[idxMin])
print('Best alpha for Misclassification Error = ', plotAlphas[idxMin])
print('')
print('Best Value for AUC = ', auc[idxMax])
print('Best alpha for AUC = ', plotAlphas[idxMax])

print('')
print('Confusion Matrices for Different Threshold Values')

#pick some points along the curve to print. There are 208 points.
#The extremes aren't useful

#Sample at 52, 104 and 156. Use the calculated values of tpr and fpr
#along with definitions and threshold values.

#Some nomenclature (e.g. see wikipedia "receiver operating curve")

#P = Positive cases
P = len(idxPos)
#N = Negative cases
N = nrow - P
#TP = True positives = tpr * P
TP = tpr[52] * P
#FN = False negatives = P - TP
FN = P - TP
#FP = False positives = fpr * N
FP = fpr[52] * N
#TN = True negatives = N - FP
TN = N - FP

print('Threshold Value = ', thresh[52])
print('TP = ', TP, 'FP = ', FP)
```

```
print('FN = ', FN, 'TN = ', TN)

TP = tpr[104] * P; FN = P - TP; FP = fpr[104] * N; TN = N - FP

print('Threshold Value = ', thresh[104])
print('TP = ', TP, 'FP = ', FP)
print('FN = ', FN, 'TN = ', TN)

TP = tpr[156] * P; FN = P - TP; FP = fpr[156] * N; TN = N - FP

print('Threshold Value = ', thresh[156])
print('TP = ', TP, 'FP = ', FP)
print('FN = ', FN, 'TN = ', TN)

Printed Output: [filename - rocksVMinesENetRegCVPrintedOutput.txt]
('Best Value of Misclassification Error = ', 0.22115384615384615)
('Best alpha for Misclassification Error = ', 0.017686244720179375)

('Best Value for AUC = ', 0.86867279650784812)
('Best alpha for AUC = ', 0.020334883589342503)

Confusion Matrices for Different Threshold Values
('Threshold Value = ', 0.37952298245219962)
('TP = ', 48.0, 'FP = ', 5.0)
('FN = ', 63.0, 'TN = ', 92.0)
('Threshold Value = ', -0.045503481125357965)
('TP = ', 85.0, 'FP = ', 20.0)
('FN = ', 26.0, 'TN = ', 77.0)
('Threshold Value = ', -0.4272522354395466)
('TP = ', 107.0, 'FP = ', 49.999999999999993)
('FN = ', 4.0, 'TN = ', 47.000000000000007)
```

交叉验证给你一个稳定的性能估计，可以据此了解实际应用系统的性能。如果交叉验证给出的性能不够好，需要努力提升。例如，可以尝试使用基扩展（如“多变量回归：预测红酒口感”对应的基扩展）。也可以查看性能表现最差的样本，看能否发现问题，如数据预处理错误，如某个特征对错误的影响最大。如果发现的错误刚好解决了你的问题，那么就应该利用整个数据集来训练部署模型。我们将在下一节介绍该过程。

构建部署用的岩石水雷分类器

对于红酒品质预测案例，完成 alpha 选择后，下一步是在全部数据集上重新训练模型，学习最佳 alpha 对应的权重系数。最佳 alpha 是指能够最小化样本外错误的参数，本例通过交叉验证进行估计。代码清单 5-5 为完成该过程的代码。

代码清单 5-5 在岩石水雷数据上训练 ElasticNet 模型的系数曲线 - rocksVMinesCoef Curves.py

```
__author__ = 'mike_bowles'
import urllib2
from math import sqrt, fabs, exp
import matplotlib.pyplot as plot
from sklearn.linear_model import enet_pathsh
from sklearn.metrics import roc_auc_score, roc_curve
import numpy

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)


#arrange data into list for labels and list of lists for attributes
xList = []


for line in data:
        #split on comma
    row = line.strip().split(",")
    xList.append(row)

#separate labels from attributes, convert attributes from
#string to numeric and convert "M" to 1 and "R" to 0

xNum = []
labels = []

for row in xList:
```

```
    lastCol = row.pop()
    if lastCol == "M":
        labels.append(1.0)
    else:
        labels.append(0.0)
    attrRow = [float(elt) for elt in row]
    xNum.append(attrRow)

#number of rows and columns in x matrix
nrow = len(xNum)
ncol = len(xNum[1])

alpha = 1.0

#calculate means and variances
xMeans = []
xSD = []
for i in range(ncol):
    col = [xNum[j][i] for j in range(nrow)]
    mean = sum(col)/nrow
    xMeans.append(mean)
    colDiff = [(xNum[j][i] - mean) for j in range(nrow)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrow)])
    stdDev = sqrt(sumSq/nrow)
    xSD.append(stdDev)
#use calculate mean and standard deviation to normalize xNum
xNormalized = []
for i in range(nrow):
    rowNormalized = [(xNum[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncol)]
    xNormalized.append(rowNormalized)

#normalize labels to center

meanLabel = sum(labels)/nrow
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] - meanLabel) \
        for i in range(nrow)])/nrow)

labelNormalized = [(labels[i] - meanLabel)/sdLabel for i in range(nrow)]
```

```
#Convert normalized labels to numpy array
Y = numpy.array(labelNormalized)

#Convert normalized attributes to numpy array
X = numpy.array(xNormalized)

alphas, coefs, _ = enet_path(X, Y,l1_ratio=0.8, fit_intercept=False,
    return_models=False)

plot.plot(alphas,coefs.T)

plot.xlabel('alpha')
plot.ylabel('Coefficients')
plot.axis('tight')
plot.semilogx()
ax = plot.gca()
ax.invert_xaxis()
plot.show()

nattr, nalpha = coefs.shape

#find coefficient ordering
nzList = []
for iAlpha in range(1,nalpha):
    coefList = list(coefs[: ,iAlpha])
    nzCoef = [index for index in range(nattr) if coefList[index] != 0.0]
    for q in nzCoef:
        if not(q in nzList):
            nzList.append(q)

#make up names for columns of X
names = ['V' + str(i) for i in range(ncol)]
nameList = [names[nzList[i]] for i in range(len(nzList))]
print("Attributes Ordered by How Early They Enter the Model")
print(nameList)
print('')
#find coefficients corresponding to best alpha value. alpha value
corresponding to normalized X and normalized Y is 0.020334883589342503

alphaStar = 0.020334883589342503
```

```
indexLTalphaStar = [index for index in range(100) if \
    alphas[index] > alphaStar]
indexStar = max(indexLTalphaStar)

#here's the set of coefficients to deploy
coefStar = list(coefs[:,indexStar])
print("Best Coefficient Values ")
print(coefStar)
print('')
#The coefficients on normalized attributes give another slightly
#different ordering

absCoef = [abs(a) for a in coefStar]

#sort by magnitude
coefSorted = sorted(absCoef, reverse=True)

idxCoefSize = [absCoef.index(a) for a in coefSorted if not(a == 0.0)]

namesList2 = [names[idxCoefSize[i]] for i in range(len(idxCoefSize))]

print("Attributes Ordered by Coef Size at Optimum alpha")
print(namesList2)

Printed Output: [filename - rocksVMinesCoefCurvesPrintedOutput.txt]
Attributes Ordered by How Early They Enter the Model
['V10', 'V48', 'V11', 'V44', 'V35', 'V51', 'V20', 'V3', 'V21', 'V45',
'V43', 'V15', 'V0', 'V22', 'V27', 'V50', 'V53', 'V30', 'V58', 'V56',
'V28', 'V39', 'V46', 'V19', 'V54', 'V29', 'V57', 'V6', 'V8', 'V7',
'V49', 'V2', 'V23', 'V37', 'V55', 'V4', 'V13', 'V36', 'V38', 'V26',
'V31', 'V1', 'V34', 'V33', 'V24', 'V16', 'V17', 'V5', 'V52', 'V41',
'V40', 'V59', 'V12', 'V9', 'V18', 'V14', 'V47', 'V42']

Best Coefficient Values
[0.082258256813766639, 0.0020619887220043702, -0.11828642590855878,
0.16633956932499627, 0.0042854388193718004, -0.0, -0.04366252474594004,
-0.07751510487942842, 0.10000054356323497, 0.0, 0.090617207036282038,
0.21210870399915693, -0.0, -0.010655386149821946, -0.0,
-0.13328659558143779, -0.0, 0.0, 0.0, 0.052814854501417867,
0.038531154796719078, 0.0035515348181877982, 0.090854714680378215,
```

```
0.030316113904025031, -0.0, 0.0, 0.0086195542357481014, 0.0, 0.0,
0.17497679257272536, -0.2215687804617206, 0.012614243827937584,
0.0, -0.0, 0.0, -0.17160601809439849, -0.080450013824209077,
0.078096790041518344, 0.022035287616766441, -0.072184409273692227,
0.0, -0.0, 0.0, 0.057018816876250704, 0.096478265685721556,
0.039917367637236176, 0.049158231541622875, 0.0, 0.22671917920123755,
-0.096272735479951091, 0.0, 0.078886784332226484, 0.0,
0.062312821755756878, -0.082785510713295471, 0.014466967172068596,
-0.074326527525632721, 0.068096475974257331,
0.070488864435477847, 0.0]

Attributes Ordered by Coef Size at Optimum alpha
['V48', 'V30', 'V11', 'V29', 'V35', 'V3', 'V15', 'V2', 'V8', 'V44',
'V49', 'V22', 'V10', 'V54', 'V0', 'V36', 'V51', 'V37', 'V7', 'V56',
'V39', 'V58', 'V57', 'V53', 'V43', 'V19', 'V46', 'V6', 'V45', 'V20',
'V23', 'V38', 'V55', 'V31', 'V13', 'V26', 'V4', 'V21', 'V1']
```

代码清单 5-5 的结构类似于代码清单 5-4，除了不包括交叉验证循环。alpha 的值是直接从代码清单 5-4 硬编码生成的，具体包括 2 个 alpha 值：一个是最小化误分类错误的 alpha 值，一个是最大化 AUC 的 alpha 值。最大化 AUC 的 alpha 值稍大并且更加保守，更加保守是因为该 alpha 比最小化分类错误的 alpha 值更靠近左边。程序打印系数显示在代码的最下边。对于 60 个系数，约有 20 个系数值为 0。对于此运行（交叉验证），l1_ratio 变量设为 0.8，这会比 l1_ratio 变量设为 1 的套索回归生成更多系数。

关于变量重要性的指标在代码的最后打印出来。衡量变量重要性的一个指标是随着 alpha 减少，进入解的变量顺序。另一个指标是根据最优解的特征系数大小得到的排序。正如红酒质量数据所讨论的，这些顺序只有当属性归一化以后才有意义。上述两种不同的变量顺序存在一定的一致性，但它们也并不完全相同。例如，变量 V48、V11、V35、V44 和 V3 在两个列表中的排名都很高，但是 V10 出现在第一个排序的开始，基于系数大小排序，V10 排名非常靠后。显然，当系数惩罚项很大时，算法只允许一个属性加入，此时 V10 很重要，但是当系数惩罚项收缩到所有属性被包含进来的对应点时，V10 属性的重要性就变得平稳，而且随着其他属性被添加进来，重要性也下降。

典型地，物体对与其同等波长级别的电磁波的反射能力最强。水雷（金属圆柱体）有长度和直径，和岩石比起来反射波波长可能较短也可能较长，性质表现更加不规则，反射的波长范围也更广。因为数据的所有属性值为正（代表功率级别），你可以预计低频波长的权重为正，高频波长的权重为负。你也可以发现这种差异如何会导致数据过拟合，即构建的模型在训练数据上表现良好，但泛化性能较差。交叉验证过程确保只要训练数据同实

际部署数据类似，模型就不会过拟合。交叉验证错误与实际部署错误应该一致，即部署时的岩石 - 水雷数比例应该与训练数据中的比例一致。

图 5-10 为使用 ElasticNet 回归模型在完整的岩石水雷数据集上的系数曲线。曲线展示了模型复杂度以及属性的相对重要性的变化情况。

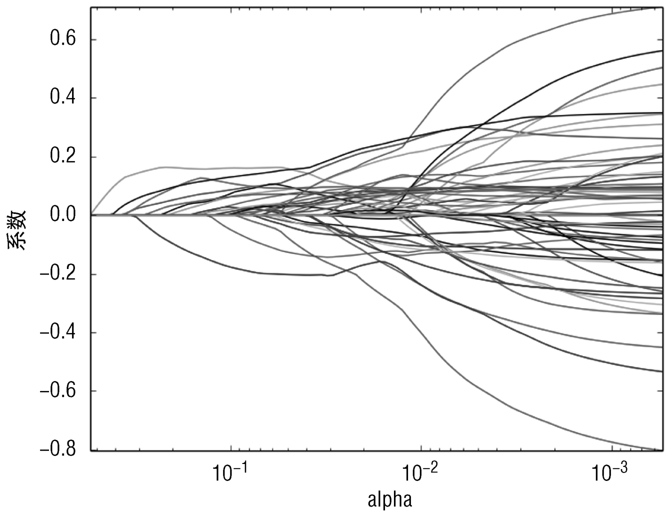


图 5-10 在岩石水雷数据上应用 ElasticNet 得到的系数曲线

正如第 4 章提到的，另一种使用惩罚线性回归模型进行分类的方法是使用惩罚逻辑回归。代码清单 5-5 为使用惩罚逻辑回归来构建岩石水雷分类器的实现代码。代码以及结果展示了两种方法的相同和不同之处。算法差异可以从迭代的结果中发现。逻辑回归方法使用特征的线性函数来计算每个训练样本属于岩石还是水雷的概率或者似然（可以到 http://en.wikipedia.org/wiki/Logistic_regression 寻找更多关于逻辑回归的背景，以及相关的偏导计算）。不含惩罚项的逻辑回归算法称作迭代重加权最小二乘法（IRLS）。名字来源于算法的本质（参照 http://en.wikipedia.org/wiki/Iteratively_reweighted_least_squares）。算法利用训练样本的概率估计来计算权重。给定权重，问题变为加权的最小二乘法问题。该过程不断迭代直到概率收敛（对应权重不再改变）。基本上，逻辑回归的 IRLS 算法相当于为第 4 章的惩罚线性回归算法（非逻辑回归）添加了另一层迭代。

将数据读入并归一化后，程序对权重及概率进行初始化，这些权重及概率是逻辑回归以及惩罚版本的逻辑回归的核心。这些概率及权重随着惩罚参数的减少同系数 β 一起计算，对于每次迭代，IRLS 字母被附加到一些变量名称上，目的是表明这些变量是和 IRLS 相关的变量。对概率估计的迭代通过循环进行，循环目的是降低 λ 以及对 β 的坐标下降进行

封装。

逻辑回归的更新细节相较于普通的惩罚线性回归（非逻辑回归）更加复杂。复杂性一方面体现在 IRLS 中权重的相关计算，每来一个样本，权重及概率计算一遍。代码使用变量 w 和 p 来代表权重和概率。算法也需要收集权重（weights）对于乘积和（Sum of products）的效果，典型乘积包括属性乘以残差以及属性值平方。上述乘积和在代码里使用 sum Wxx 变量来定义，具体地，sum Wxx 是一个 list 对象，每个元素为权重乘以对应属性值平方。复杂性的另一方面体现在残差是标签、概率以及属性值以及相关系数（β）的函数。

代码生成了变量重要性顺序及系数曲线，目的是与使用非逻辑回归生成的变量顺序及曲线进行比较。逻辑转换使得直接进行系数比较存在问题，因为逻辑函数对应非线性变化。普通线性回归以及逻辑回归都会生成一个系数向量，然后乘以属性值，最后和阈值比较。阈值设置相对次要，因为它可以在训练完成后来确定。所以 β 的整体尺度和各个特征值的相对大小比较并不重要。一种判断相对大小的方法是查看两种方法引入新变量的顺序。通过比较代码清单 5-5 和代码清单 5-4 的打印结果，两种方法对排序最高的前 8 个属性的判断是一致的。对于接下来的 8 个变量，有 7 个变量同时出现在两个列表中，尽管也有很小的区别。接下来的 8 个变量也几乎相同。这说明两种方法对属性重要性顺序的判断非常一致。

另一个问题是哪种方法性能更好，回答这个问题需要对惩罚逻辑回归运行交叉验证，好在你已经有相关的工具及代码来开展工作。代码清单 5-6 没有从速度角度进行优化，但在岩石水雷问题上运行应该不会花费太多时间。

代码清单 5-6　在岩石水雷数据上训练惩罚逻辑回归模型 - rocksVMinesGlmnet.py

```
__author__ = 'mike_bowles'
import urllib2
import sys
from math import sqrt, fabs, exp
import matplotlib.pyplot as plot

def S(z,gamma):
    if gamma >= fabs(z):
        return 0.0
    if z > 0.0:
        return z - gamma
    else:
        return z + gamma

def Pr(b0,b,x):
```

```
    n = len(x)
    sum = b0
    for i in range(n):
        sum += b[i]*x[i]
        if sum < -100: sum = -100
    return 1.0/(1.0 + exp(-sum))


#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []


for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

#separate labels from attributes, convert from attributes from string
#to numeric and convert "M" to 1 and "R" to 0

xNum = []
labels = []

for row in xList:
    lastCol = row.pop()
    if lastCol == "M":
        labels.append(1.0)
    else:
        labels.append(0.0)
    attrRow = [float(elt) for elt in row]
    xNum.append(attrRow)

#number of rows and columns in x matrix
nrow = len(xNum)
ncol = len(xNum[1])
```

```
alpha = 0.8
#calculate means and variances
xMeans = []
xSD = []
for i in range(ncol):
    col = [xNum[j][i] for j in range(nrow)]
    mean = sum(col)/nrow
    xMeans.append(mean)
    colDiff = [(xNum[j][i] - mean) for j in range(nrow)]
    sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrow)])
    stdDev = sqrt(sumSq/nrow)
    xSD.append(stdDev)

#use calculate mean and standard deviation to normalize xNum
xNormalized = []
for i in range(nrow):
    rowNormalized = [(xNum[i][j] - xMeans[j])/xSD[j] \
        for j in range(ncol)]
    xNormalized.append(rowNormalized)

#Do Not Normalize labels but do calculate averages
meanLabel = sum(labels)/nrow
sdLabel = sqrt(sum([(labels[i] - meanLabel) * (labels[i] - meanLabel) \
    for i in range(nrow)])/nrow)

#initialize probabilities and weights
sumWxr = [0.0] * ncol
sumWxx = [0.0] * ncol
sumWr = 0.0
sumW = 0.0

#calculate starting points for betas
for iRow in range(nrow):
    p = meanLabel
    w = p * (1.0 - p)
    #residual for logistic
    r = (labels[iRow] - p) / w
    x = xNormalized[iRow]
    sumWxr = [sumWxr[i] + w * x[i] * r for i in range(ncol)]
    sumWxx = [sumWxx[i] + w * x[i] * x[i] for i in range(ncol)]
    sumWr = sumWr + w * r
```

```
        sumW = sumW + w
    avgWxr = [sumWxr[i]/nrow for i in range(ncol)]
    avgWxx = [sumWxx[i]/nrow for i in range(ncol)]

    maxWxr = 0.0
    for i in range(ncol):
        val = abs(avgWxr[i])
        if val > maxWxr:
            maxWxr = val

    #calculate starting value for lambda
    lam = maxWxr/alpha

    #this value of lambda corresponds to beta = list of 0's
    #initialize a vector of coefficients beta
    beta = [0.0] * ncol
    beta0 = sumWr/sumW

    #initialize matrix of betas at each step
    betaMat = []
    betaMat.append(list(beta))

    beta0List = []
    beta0List.append(beta0)

    #begin iteration
    nSteps = 100
    lamMult = 0.93 #100 steps gives reduction by factor of 1000 in lambda
                   #(recommended by authors)
    nzList = []
    for iStep in range(nSteps):
        #decrease lambda
        lam = lam * lamMult


        #Use incremental change in betas to control inner iteration


        #set middle loop values for betas = to outer values
        #values are used for calculating weights and probabilities
        #inner values are used for calculating penalized regression updates
```

```
#take pass through data to calculate averages over data required
#for iteration
#initilize accumulators

betaIRLS = list(beta)
beta0IRLS = beta0
distIRLS = 100.0
#Middle loop to calculate new betas with fixed IRLS weights and
#probabilities
iterIRLS = 0
while distIRLS > 0.01:
    iterIRLS += 1
    iterInner = 0.0

    betaInner = list(betaIRLS)
    beta0Inner = beta0IRLS
    distInner = 100.0
    while distInner > 0.01:
        iterInner += 1
        if iterInner > 100: break

        #cycle through attributes and update one-at-a-time
        #record starting value for comparison
        betaStart = list(betaInner)
        for iCol in range(ncol):

            sumWxr = 0.0
            sumWxx = 0.0
            sumWr = 0.0
            sumW = 0.0

            for iRow in range(nrow):
                x = list(xNormalized[iRow])
                y = labels[iRow]
                p = Pr(beta0IRLS, betaIRLS, x)
                if abs(p) < 1e-5:
                    p = 0.0
                    w = 1e-5
                elif abs(1.0 - p) < 1e-5:
                    p = 1.0
```

```
                    w = 1e-5
                else:
                    w = p * (1.0 - p)

                z = (y - p) / w + beta0IRLS + sum([x[i] *
                    betaIRLS[i] for i in range(ncol)])
                r = z - beta0Inner - sum([x[i] * betaInner[i]
                    for i in range(ncol)])
                sumWxr += w * x[iCol] * r
                sumWxx += w * x[iCol] * x[iCol]
                sumWr += w * r
                sumW += w

            avgWxr = sumWxr / nrow
            avgWxx = sumWxx / nrow

            beta0Inner = beta0Inner + sumWr / sumW
            uncBeta = avgWxr + avgWxx * betaInner[iCol]
            betaInner[iCol] = S(uncBeta, lam * alpha) / (avgWxx +
                lam * (1.0 - alpha))
        sumDiff = sum([abs(betaInner[n] - betaStart[n]) \
            for n in range(ncol)])
        sumBeta = sum([abs(betaInner[n]) for n in range(ncol)])
        distInner = sumDiff/sumBeta
    #print number of steps for inner and middle loop convergence
    #to monitor behavior
    #print(iStep, iterIRLS, iterInner)

    #if exit inner while loop, then set betaMiddle = betaMiddle
    #and run through middle loop again.

    #Check change in betaMiddle to see if IRLS is converged
    a = sum([abs(betaIRLS[i] - betaInner[i]) for i in range(ncol)])
        b = sum([abs(betaIRLS[i]) for i in range(ncol)])
        distIRLS = a / (b + 0.0001)
        dBeta = [betaInner[i] - betaIRLS[i] for i in range(ncol)]
        gradStep = 1.0
        temp = [betaIRLS[i] + gradStep * dBeta[i] for i in range(ncol)]
        betaIRLS = list(temp)

    beta = list(betaIRLS)
```

```
        beta0 = beta0IRLS
        betaMat.append(list(beta))
        beta0List.append(beta0)

        nzBeta = [index for index in range(ncol) if beta[index] != 0.0]
        for q in nzBeta:
            if not(q in nzList):
                nzList.append(q)

#make up names for columns of xNum
names = ['V' + str(i) for i in range(ncol)]
nameList = [names[nzList[i]] for i in range(len(nzList))]

print("Attributes Ordered by How Early They Enter the Model")
print(nameList)
for i in range(ncol):
    #plot range of beta values for each attribute
    coefCurve = [betaMat[k][i] for k in range(nSteps)]
    xaxis = range(nSteps)
    plot.plot(xaxis, coefCurve)

plot.xlabel("Steps Taken")
plot.ylabel("Coefficient Values")
plot.show()

Printed Output: [filename - rocksVMinesGlmnetPrintedOutput.txt]

Attributes Ordered by How Early They Enter the Model
['V10', 'V48', 'V11', 'V44', 'V35', 'V51', 'V20', 'V3', 'V50', 'V21',
'V43', 'V47', 'V15', 'V27', 'V0', 'V22', 'V36', 'V30', 'V53', 'V56',
'V58', 'V6', 'V19', 'V28', 'V39', 'V49', 'V7', 'V23', 'V54', 'V8',
'V14', 'V2', 'V29', 'V38', 'V57', 'V45', 'V13', 'V32', 'V31', 'V42',
'V16', 'V37', 'V59', 'V52', 'V25', 'V18', 'V1', 'V33', 'V4', 'V55',
'V17', 'V46', 'V26', 'V12', 'V40', 'V34', 'V5', 'V24', 'V41', 'V9']
```

图 5-11 展示了使用惩罚逻辑回归的岩石水雷系数曲线。正如所标记的，系数尺度与普通惩罚线性回归尺度不同，因为两种方法使用的逻辑函数不同。普通回归尝试为目标 0 和 1 拟合一条直线，逻辑回归通过拟合一条“对数奇数比”的直线来预测类别成员的概率。假设 p 是样本属于水雷类别的概率，然后奇数比等于 $p/1-p$。对数奇数比是奇数比的自然对数。只要 p 的范围是 0 ～ 1，对数奇数比的范围就是 $-\infty \sim \infty$。对数奇数比非常大并

且为正表明预测样本属于水雷类的结论非常确定。负的较大的数值对应于岩石类别。

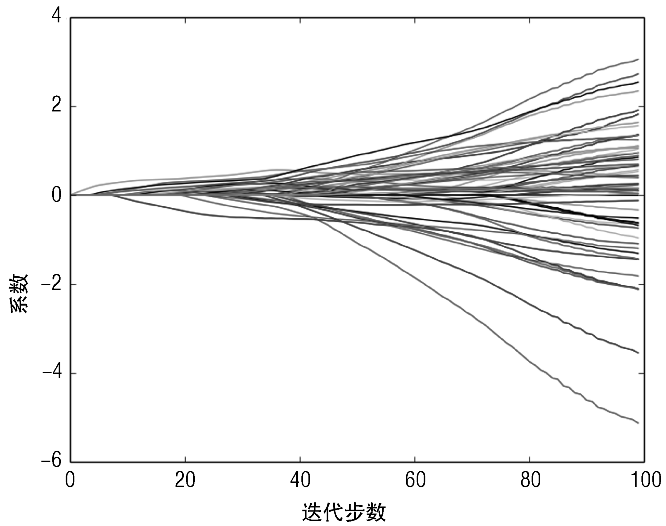


图 5-11 在岩石水雷数据上训练 ElasticNet 惩罚逻辑回归得到的系数曲线

因为两种方法的预测差异非常大，所以生成的预测尺度不同，对应属性的系数也不同。但正如两种程序打印输出所表现的，变量出现在解的顺序非常类似，这可以通过系数曲线中出现的前几个属性印证得到。

5.4 多类别分类 - 分类犯罪现场的玻璃样本

上一节看到的岩石水雷问题称作 2 分类问题，即预测只能取两种可能值中的 1 种（即返回的声纳是来自于岩石，还是水雷的反射？）如果预测标签不止两个值，那么问题称作多类别分类问题。本节使用惩罚线性回归来解决玻璃样本分类问题，正如第 2 章讨论的，玻璃样本包含 9 个物理化学指标（基于其化学组成的折射率）、6 种类型玻璃，共 214 个样本。问题是使用物理化学指标来确定给定样本属于 6 种类型的哪一种。实际应用于犯罪或者车祸现场的法医分析。数据来源于 UCI 数据集，相关页面提供了一篇使用支持向量机求解问题的论文。在阅读完解决该问题的代码后，本节将对惩罚线性回归方法以及支持向量机的方法进行性能比较。

代码清单 5-7 为解决该问题的代码。

代码清单 5-7 使用惩罚线性回归的多类别分类 - 分类犯罪现场的玻璃样本 - glass ENetRegCV.py

```
import urllib2
from math import sqrt, fabs, exp
import matplotlib.pyplot as plot
```

```
from sklearn.linear_model import enet_path
from sklearn.metrics import roc_auc_score, roc_curve
import numpy

target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/glass/glass.data")
data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []
for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

names = ['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Type']

#Separate attributes and labels
xNum = []
labels = []

for row in xList:
    labels.append(row.pop())
    l = len(row)
    #eliminate ID
    attrRow = [float(row[i]) for i in range(1, l)]
    xNum.append(attrRow)

#number of rows and columns in x matrix
nrow = len(xNum)
ncol = len(xNum[1])

#create one versus all label vectors
#get distinct glass types and assign index to each
yOneVAll = []
labelSet = set(labels)
labelList = list(labelSet)
labelList.sort()
nlabels = len(labelList)
for i in range(nrow):
    yRow = [0.0]*nlabels
```

```
        index = labelList.index(labels[i])
        yRow[index] = 1.0
        yOneVAll.append(yRow)

    #calculate means and variances
    xMeans = []
    xSD = []
    for i in range(ncol):
        col = [xNum[j][i] for j in range(nrow)]
        mean = sum(col)/nrow
        xMeans.append(mean)
        colDiff = [(xNum[j][i] - mean) for j in range(nrow)]
        sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrow)])
        stdDev = sqrt(sumSq/nrow)
        xSD.append(stdDev)

    #use calculate mean and standard deviation to normalize xNum
    xNormalized = []
    for i in range(nrow):
        rowNormalized = [(xNum[i][j] - xMeans[j])/xSD[j] \
            for j in range(ncol)]
        xNormalized.append(rowNormalized)

    #normalize y's to center
    yMeans = []
    ySD = []
    for i in range(nlabels):
        col = [yOneVAll[j][i] for j in range(nrow)]
        mean = sum(col)/nrow
        yMeans.append(mean)
        colDiff = [(yOneVAll[j][i] - mean) for j in range(nrow)]
        sumSq = sum([colDiff[i] * colDiff[i] for i in range(nrow)])
        stdDev = sqrt(sumSq/nrow)
        ySD.append(stdDev)

    yNormalized = []
    for i in range(nrow):
        rowNormalized = [(yOneVAll[i][j] - yMeans[j])/ySD[j] \
            for j in range(nlabels)]
        yNormalized.append(rowNormalized)
```

```
#number of cross-validation folds
nxval = 10
nAlphas=200
misClass = [0.0] * nAlphas

for ixval in range(nxval):
    #Define test and training index sets
    idxTest = [a for a in range(nrow) if a%nxval == ixval%nxval]
    idxTrain = [a for a in range(nrow) if a%nxval != ixval%nxval]

    #Define test and training attribute and label sets
    xTrain = numpy.array([xNormalized[r] for r in idxTrain])
    xTest = numpy.array([xNormalized[r] for r in idxTest])
    yTrain = [yNormalized[r] for r in idxTrain]
    yTest = [yNormalized[r] for r in idxTest]
    labelsTest = [labels[r] for r in idxTest]

    #build model for each column in yTrain
    models = []
    lenTrain = len(yTrain)
    lenTest = nrow - lenTrain
    for iModel in range(nlabels):
        yTemp = numpy.array([yTrain[j][iModel]
            for j in range(lenTrain)])
        models.append(enet_path(xTrain, yTemp,l1_ratio=1.0,
            fit_intercept=False, eps=0.5e-3, n_alphas=nAlphas ,
            return_models=False))

    for iStep in range(1,nAlphas):
        #Assemble the predictions for all the models, find largest
        #prediction and calc error
        allPredictions = []
        for iModel in range(nlabels):
            _, coefs, _ = models[iModel]
            predTemp = list(numpy.dot(xTest, coefs[:,iStep]))
            #un-normalize the prediction for comparison
            predUnNorm = [(predTemp[j]*ySD[iModel] + yMeans[iModel]) \
                for j in range(len(predTemp))]
            allPredictions.append(predUnNorm)
```

```
        predictions = []
        for i in range(lenTest):
            listOfPredictions = [allPredictions[j][i] \
                for j in range(nlabels) ]
            idxMax = listOfPredictions.index(max(listOfPredictions))
            if labelList[idxMax] != labelsTest[i]:
                misClass[iStep] += 1.0

misClassPlot = [misClass[i]/nrow for i in range(1, nAlphas)]

plot.plot(misClassPlot)

plot.xlabel("Penalty Parameter Steps")
plot.ylabel(("Misclassification Error Rate"))
plot.show()
```

代码第一部分用于从 UCI 网站读取数据，同时将属性与标签分离开。属性以常见的方式进行归一化。使用“一对所有”进行多类别分类的方法在标签处理上存在一些差异。“一对所有”方法针对每个标签生成一个要预测的标签向量。回归和 2 分类问题只包含一个标签向量，而玻璃问题存在 6 个标签，所以对应生成 6 个标签向量。这么做的原因是：如果将数据点划分为两组，一个平面就可以做到。如果问题是将数据点划分为 6 组，那么需要多个平面。

“一对所有”训练的分类器个数与标签个数相同。这些不同分类器针对不同的标签向量训练得到。代码清单 5-7 展示了如何基于给定问题的原始类别标签来构建新的标签。方法非常类似于第 4 章看到的方法，用于将类别变量转换为数值变量。代码清单使用 Python 集合来提取不同标签，按从小到大的顺序排列（不是完全必须的，但对保持代码条理性很有帮助），形成了一个标签列表；对于每一行数据，使用一个新的标签行来表示原始标签，如果原始标签取第一个标签值，那么新标签行的第一列为 1 ；如果原始标签取第 2 个标签值，那么新标签行的第 2 列为 1，以此类推。你可以看到为什么被称作“一对所有”了。第一列的标签会产生一个 2 分类器，用于预测样本是否取第 1 个标签值。6 个分类器的每一个都需要做类似的二分类决策。

继续往下，沿着我们熟悉的线路，代码构建一个交叉验证循环。一个较小的差别是原始标签也被切分为测试集，用于后续误分类错误的度量。模型训练存在一个明显差异，每次交叉验证会训练 6 个模型，这些模型被存储在一个列表中便于后续使用。enet_path 的调用也有一些变化。一个是 eps 参数被设置为默认 1e-3 的一半，eps 作为惩罚参数值，可以任意设置。第 4 章提到使用坐标下降方法通过减少惩罚参数来选择 eps。实际上，eps

参数告诉算法进行到什么程度停止。eps 输入是惩罚参数的结束值除以起始值的比例。参数 n_alphas 控制参数更新步数。注意如果步数太大，可能算法不会收敛。如果算法收敛失败会给出警告信息。可以接着稍微调大 eps，使惩罚参数不会在每步大幅下降，或者可以调大 n_alphas 来增加步长，使得每个单独步长变小。

另一个注意事项是是否绘制了曲线的完整内容。图 5-12 展示了最小值非常接近于图的右侧边缘。此时要尽可能地往外延升曲线，以确保最小值被纳入视线范围。减少 eps 会继续向右绘制曲线。

训练得到的 6 个模型用于生成 6 个预测结果。代码然后检测 6 个预测结果的哪个结果有最大的数值输出，最终选择输出最大的预测。预测值接着和实际值比较，并对错误进行累加。

图 5-12 展示了误分类错误随惩罚参数减少的变化情况。图示展现了从左边的最简单模型开始，向右推进，错误最小值显著下降。误分类错误的最小值大约为 35%。该值要好于基于线性核的支持向量机。论文使用非线性核获得的误分类错误约为 35%，对于某些线性核，误分类错误低至 30%。在支持向量机中使用非线性核约等价于使用基扩展（正如本章前面红酒品质预测的例子）。基扩展在红酒品质预测中并不有效，但使用非线性核带来的性能提升预示着在玻璃分类问题也可能取得好的效果。

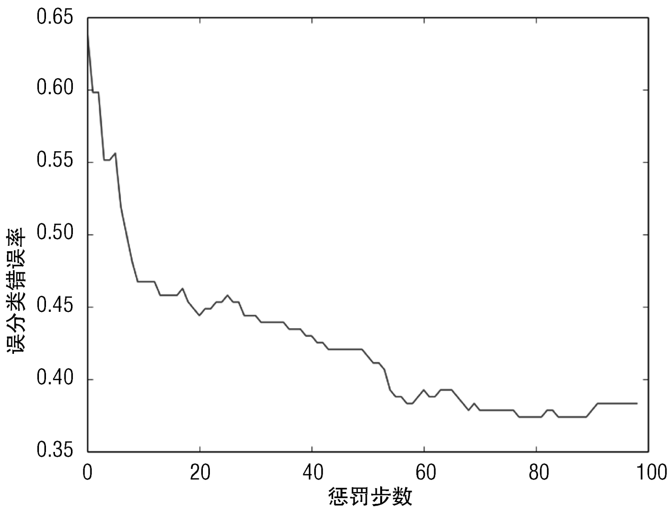


图 5-12 对玻璃分类问题使用惩罚线性回归得到的误分类错误率

小结

本章展示了对预测建模问题使用惩罚回归以及一些通用工具的案例，以及实际应用中

经常会遇到的几种不同类型的问题。这些问题包括回归、二分类以及多类别分类问题。本章使用基于 Python 的不同版本的惩罚回归函数来解决这些问题。此外，本章还展示了几种工具的使用方法。这些工具包括对类别变量的编码,使用二分器来解决多类别分类问题，对线性方法进行扩展来预测属性及输出之间的非线性关系。

本章还介绍了模型性能评价方法。回归问题最容易评估，因为它的错误可以表示为数值。分类问题也可以概括进来。我们看到分类性能可以被量化为误分类错误率、接收曲线的曲线下面积，以及经济代价。应该挑选最能反映实际目标的性能指标，这些目标包括商业目标、科学目标等。

参考文献

1. P. Cortez , A. Cerdeira , F. Almeida , T. Matos , and J. Reis . (2009). Modeling wine preferences by data mining from physicochemical properties . Decision Support Systems , Elsevier , 47(4): 547 – 553.

2. T. Hastie , R. Tibshirani , and J. Friedman . (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. Springer-Verlag , New York .

3. J. Friedman , T. Hastie , and R. Tibshirani . (2010). Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software , 33(1).

4. K. Bache and M. Lichman . (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science . http://archive.ics.uci.edu/ml.

第 6 章

集成方法

集成方法来源于下述观察：如果模型之间近似相互独立，则多个模型联合的性能要优于单个模型的。如果一个分类器以 55% 的概率可以给出正确的结果，这样的分类器只能说是中等水平，但是如果拥有 100 个这样的分类器，则大多数分类器的结果都正确的概率可以上升到 82%。（在谷歌搜索“累积二项式概率分布”可以看到详细的信息，你也可以尝试其他的概率值看看会得到什么样的结果。）

一种获取近似相互独立的多个模型的方法就是使用不同的机器学习算法。例如，可以利用支持向量机（SVM）、线性回归、k 最近邻、二元决策树等等。但是这种方法很难产生大量的模型，而且这个过程冗长乏味，因为不同的模型有不同参数，需要分别调参，而且每个模型对输入数据的要求也不同。因此每个模型需要分别编码。这就远远不能适应于需要成百上千个模型的场景（下面会遇到）。

因此，集成方法的关键是开发出一种可以生成大量近似独立的模型的方法，然后把它们集成起来。在本章将学到最流行的方法是如何做的，并了解最流行的集成方法的工作原理。本章概述算法的基本架构，用 Python 展示算法的有效性，以加深对算法原理的理解。

集成方法是由两层算法组成的层次架构。底层的算法叫作基学习器（base learner）。基学习器是单个机器学习算法，这些算法后续会被集成到一个集成方法中。本章主要使用二元决策树作为基学习器。上层算法通过对这些基学习器做巧妙的处理，使其模型近似相对独立。那么同一算法如何产生不同的模型？目前广泛使用的上层算法主要有：投票（bagging）、提升（boosting）和随机森林（random forests）。严格地讲，随机森林实际上是上层算法和修改后的二元决策树的组合，在“随机森林”小节可以看到详细的内容。

有很多算法都可以用作基学习器，如二元决策树、支持向量机等，但从实用角度二元决策树的应用最为广泛。它们广泛应用于开源和商业的软件包中，这些软件包都可以应用到项目中。集成方法包含成千上万的二元决策树，集成方法的很多特性都源自二元决策树。所以本章以二元决策树开始介绍。

6.1 二元决策树

二元决策树就是基于属性做一系列的二元（是 / 否）决策。每次决策对应于从两种可能性中选择一个。每次决策后，要么引出另外一个决策，要么生成最终的结果。一个实际训练决策树的例子有助于加强对这个概念的理解。了解了训练后的决策树是什么样的，就学会了决策树的训练过程。

代码清单 6-1 为使用 Scikitlearn 的 DecisionTreeRegressor 工具包针对红酒口感数据构建二元决策树的代码。图 6-1 为代码清单 6-1 生成的决策树。

代码清单 6-1 构建一个决策树预测红酒口感 -winTree.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.externals.six import StringIO
from math import sqrt
import matplotlib.pyplot as plot

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
```

```
            row.pop()
            #convert row to floats
            floatRow = [float(num) for num in row]
            xList.append(floatRow)

    nrows = len(xList)
    ncols = len(xList[0])

    wineTree = DecisionTreeRegressor(max_depth=3)

    wineTree.fit(xList, labels)

    with open("wineTree.dot", 'w') as f:
        f = tree.export_graphviz(wineTree, out_file=f)
    #Note: The code above exports the trained tree info to a
    #Graphviz "dot" file.
    #Drawing the graph requires installing GraphViz and the running the
    #following on the command line
    #dot -Tpng wineTree.dot -o wineTree.png
    # In Windows, you can also open the .dot file in the GraphViz
    #gui (GVedit.exe)]
```

图 6-1 为针对红酒数据的训练结果，即一系列的决策。决策树框图显示了一系列的方框，这些方框称作节点（nodes）。有两类节点，一种针对问题输出“是”或者“否”，另外一种是终止节点，输出针对样本的预测结果，并终止整个决策的过程。终止节点也叫作叶子节点（leaf）。在图 6-1 中，终止节点处在框图底部，它们下面没有分支或者进一步的决策节点。

6.1.1 如何利用二元决策树进行预测

当一个观察(或一行数据)被传送到一个非终止节点时,此行数据要回答此节点的问题。如果回答“是”，则该行数据进入节点下面的左侧节点。如果回答”否“，则此行数据进入节点下面的右侧节点。该过程持续进行，直到到达一个终止节点（即叶子节点）,叶子节点给该行数据分配预测值。叶子节点分配的预测值是所有到达此节点的训练数据结果的均值。

尽管此决策树的第二个决策层在两个分支中都考虑了变量 X[9],这两个决策也可以是针对不同属性所做的判断（可以参看第三个决策层的例子）。

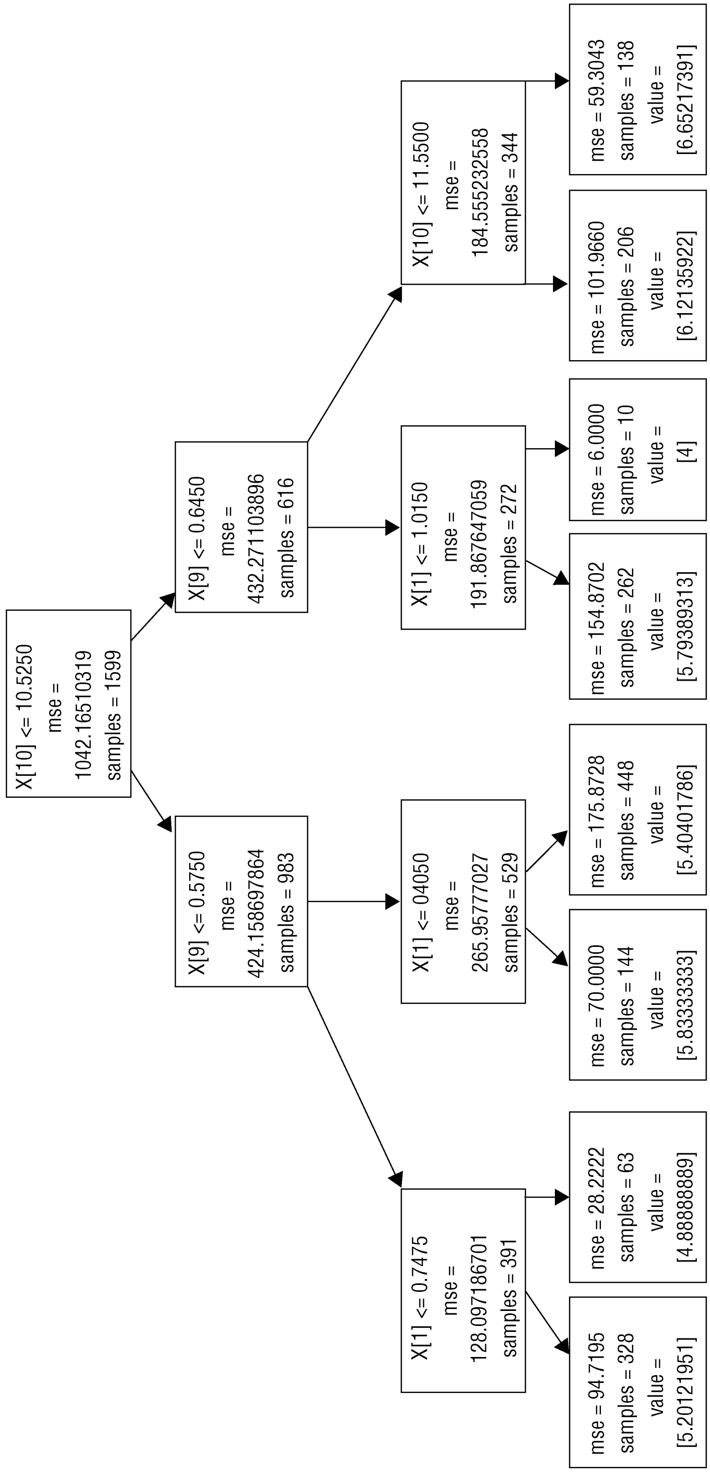


图 6-1 确定红酒口感的决策树

最上面的节点又叫根节点（root node）。这个节点提出的问题是“X[10]<=10.525”。在二元决策树中，越是重要的变量越早用来分割数据（越接近决策树的顶端），因此决策树认为变量 X[10]，也就是酒精含量属性很重要。这点决策树与第 5 章的惩罚线性回归是一致的。第 5 章“用惩罚线性方法构建预测模型”也认为酒精含量是决定红酒口感最重要的属性。

图 6-1 所示决策树的深度为 3。决策树的深度定义为从上到下遍历树的最长路径（所经过的决策的数目）。在“决策树的训练等价于分割点的选择”小节的关于训练的讨论中，可以看到没有理由要求到达终止节点的所有路径具有相同的长度（见图 6-1）。

现在已经知道一个训练好的决策树是什么样的，也看到了如何使用一个决策树来进行预测。下面介绍如何训练决策树。

6.1.2 如何训练一个二元决策树

了解如何训练决策树最简单的方法就是通过一个具体的例子。代码清单 6-2 为给定一个实数属性如何预测一个实数标签的例子。数据集在代码中产生（也叫作合成数据）。生成过程是把 −0.5 ～ +0.5 等分成 100 份，单一实数属性 x 就是这些等分数。标签 y 等于 x 加上随机噪声。

代码清单 6-2 简单回归问题的决策树训练 -simpleTree.py

```
__author__ = 'mike-bowles'

import numpy
import matplotlib.pyplot as plot
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.externals.six import StringIO

#Build a simple data set with y = x + random
nPoints = 100

#x values for plotting
xPlot = [(float(i)/float(nPoints) - 0.5) for i in range(nPoints + 1)]

#x needs to be list of lists.
x = [[s] for s in xPlot]

#y (labels) has random noise added to x-value
#set seed
numpy.random.seed(1)
```

```
y = [s + numpy.random.normal(scale=0.1) for s in xPlot]

plot.plot(xPlot,y)
plot.axis('tight')
plot.xlabel('x')
plot.ylabel('y')
plot.show()

simpleTree = DecisionTreeRegressor(max_depth=1)
simpleTree.fit(x, y)

#draw the tree
with open("simpleTree.dot", 'w') as f:
    f = tree.export_graphviz(simpleTree, out_file=f)

#compare prediction from tree with true values

yHat = simpleTree.predict(x)

plot.figure()
plot.plot(xPlot, y, label='True y')
plot.plot(xPlot, yHat, label='Tree Prediction ', linestyle='--')
plot.legend(bbox_to_anchor=(1,0.2))
plot.axis('tight')
plot.xlabel('x')
plot.ylabel('y')
plot.show()

simpleTree2 = DecisionTreeRegressor(max_depth=2)
simpleTree2.fit(x, y)

#draw the tree
with open("simpleTree2.dot", 'w') as f:
    f = tree.export_graphviz(simpleTree2, out_file=f)

#compare prediction from tree with true values

yHat = simpleTree2.predict(x)

plot.figure()
```

```
plot.plot(xPlot, y, label='True y')
plot.plot(xPlot, yHat, label='Tree Prediction ', linestyle='--')
plot.legend(bbox_to_anchor=(1,0.2))
plot.axis('tight')
plot.xlabel('x')
plot.ylabel('y')
plot.show()

#split point calculations - try every possible split point to
#find the best one
sse = []
xMin = []
for i in range(1, len(xPlot)):
    #divide list into points on left and right of split point
    lhList = list(xPlot[0:i])
    rhList = list(xPlot[i:len(xPlot)])

    #calculate averages on each side
    lhAvg = sum(lhList) / len(lhList)
    rhAvg = sum(rhList) / len(rhList)

    #calculate sum square error on left, right and total
    lhSse = sum([(s - lhAvg) * (s - lhAvg) for s in lhList])
    rhSse = sum([(s - rhAvg) * (s - rhAvg) for s in rhList])

    #add sum of left and right to list of errors

    sse.append(lhSse + rhSse)
    xMin.append(max(lhList))

plot.plot(range(1, len(xPlot)), sse)
plot.xlabel('Split Point Index')
plot.ylabel('Sum Squared Error')
plot.show()

minSse = min(sse)
idxMin = sse.index(minSse)
print(xMin[idxMin])

#what happens if the depth is really high?
```

```
simpleTree6 = DecisionTreeRegressor(max_depth=6)
simpleTree6.fit(x, y)

#too many nodes to draw the tree
#with open("simpleTree2.dot", 'w') as f:
# f = tree.export_graphviz(simpleTree6, out_file=f)

#compare prediction from tree with true values

yHat = simpleTree6.predict(x)

plot.figure()
plot.plot(xPlot, y, label='True y')
plot.plot(xPlot, yHat, label='Tree Prediction ', linestyle='-')
plot.legend(bbox_to_anchor=(1,0.2))
plot.axis('tight')
plot.xlabel('x')
plot.ylabel('y')
plot.show()
```

图 6-2 为属性 *x* 和标签 *y* 的关系图。正如预期，*y* 值大致上一直跟随 *x* 值变化，但是有些随机的小扰动。

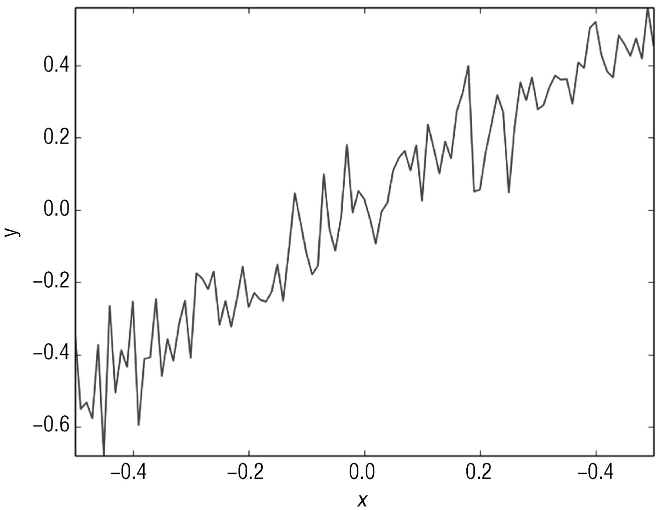


图 6-2 标签与属性的关系图

6.1.3 决策树的训练等同于分割点的选择

代码清单 6-2 的第一步是运行 scikitlearn 的 regression tree 包，并指定决策树的深度为 1。此处理过程的结果如图 6-3 所示。图 6-3 为深度为 1 的决策树的框图。深度为 1 的树又叫作桩（stumps）。在根节点的决策就是将属性值与 −0.075 比较。这个值叫作分割点（split point），因为它把数据分割成两部分。由根节点发散出去的两个方框可知，101 个实例中有 43 个到了根节点的左分支，剩下的 58 个实例到了根节点的右分支。如果属性值小于分割点，则此决策树的预测值就是方框里指明的值，大约就是 −0.302。

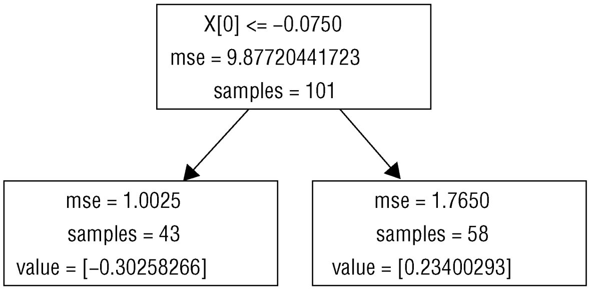


图 6-3 一个简单问题的解：深度为 1 的决策树的框图

分割点的选择如何影响预测效果

审视决策树的另一个方法就是将预测值与真实的标签值进行对比。这个简单的合成数据只有一个属性，由决策树产生的预测值一直跟随着实际的标签值，从中也能看出这个简单的决策树的训练是如何完成的。如图 6-4 所示，预测的值是基于一个简单的判断方法。预测值实际上是属性值的阶梯函数。这个“阶梯”就发生在分割点。

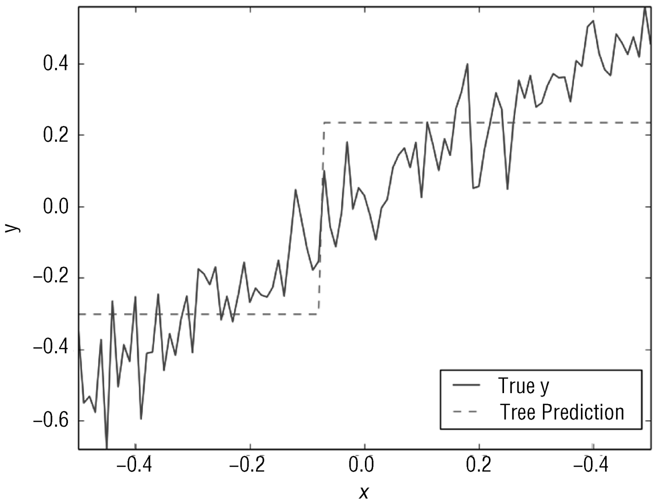


图 6-4 预测值与实际值的比较

分割点选择算法

这个简单的决策树需要确定 3 个变量：分割点的值、分割后生成的两组数据的预测值。决策树的训练过程就是要完成这个任务。下面介绍上述目标如何达到。训练此决策树的目标是使预测值的误差平方最小。首先假设分割点已经确认。一旦给定分割点，分配给两个组的预测值就可以确定下来。分配的值就是使均方误差最小的那个值。那么剩下的问题就是如何确定分割点的值。代码清单 6-2 有一小段代码用来确定分割点。这个过程是尝试每一个可能的分割点，然后把数据分成 2 组，取每组数值的均值作为分配的预测值，然后计算相应的误差平方和。

图 6-5 展示了误差平方和作为分割点的函数是如何变化的。大概在数据的中心，可以明确地取到最小的误差平方。训练一个决策树需要穷尽地搜索所有可能的分割点来确定哪个值可以使误差平方和最小化。这是这个简单的例子需要注意的地方。

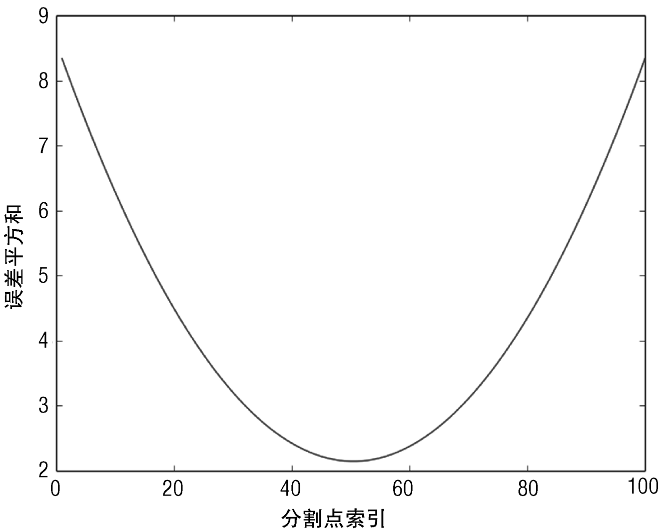


图 6-5 每个可能的分割点对应的误差平方和

多变量决策树的训练 - 选择哪个属性进行分割

如果问题含有多个属性该怎么办？算法会对所有的属性检查所有可能的分割点，对每个属性找到误差平方和最小的分割点，然后找到哪个属性对应的误差平方和最小。

在训练决策树的过程中，每个计算周期都要对分割点进行计算。同样地，训练基于决策树的集成算法时，每个周期也要对分割点进行计算。如果属性没有重复值，每个数据点对应的属性值都要作为分割点进行测试（则分割点的测试次数等于数据点数目减 1）。

随着数据规模的增大，分割点的计算量也成比例增加。测试的分割点彼此可能非常近。因此设计针对大规模数据的算法时，分割点的检测通常要比原始数据的粒度粗糙得多。论文“PLANET:Massively Parallel Learning of Tree Ensembles with MapReduce”提出一种方法，是谷歌工程师针对大规模数据集构建决策树时采用的方法，他们使用决策树来实现梯度提升（gradient boosting）算法（本章将会学到该集成方法）。

通过递归分割获得更深的决策树

代码清单 6-2 展示了当决策树深度从 1 增加到 2 时，预测曲线会发生什么变化。预测曲线如图 6-6 所示。决策树的框图如图 6-7 所示。深度为 1 的决策树只有一步，这个预测曲线有 3 步。第 2 决策层分割点的确定与第 1 个分割点的方法完全一样。决策树的每个节点处理基于上个分割点生成的数据子集。每个节点中分割点的选择是使下面 2 个节点的误差平方和最小。图 6-6 的曲线非常接近一个实际的阶梯函数曲线。决策树深度的增加意味着更细小的步长、更高的保真度（准确性）。但是如果这个过程无限地继续下去会怎样？

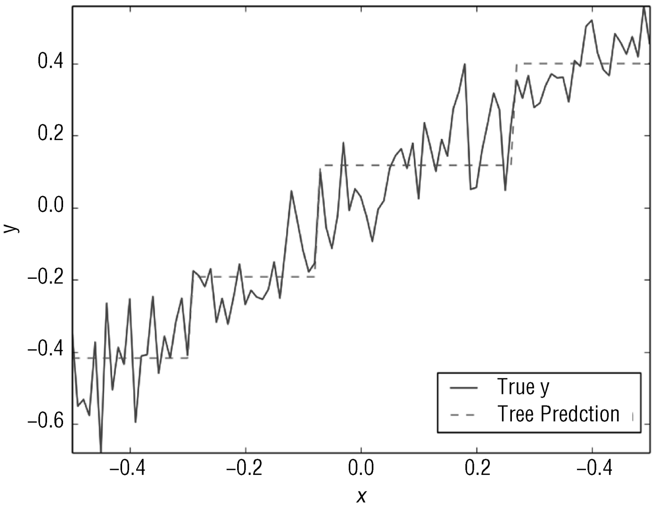


图 6-6 深度为 2 的决策树的预测曲线

随着分割的继续，决策树深度增加，最深节点包含的数据（实例数）会减少。这将导致在达到特定的深度之前，这种分割就终止了。如果决策树的节点只有一个数据实例，就不需要分割了。决策树训练算法通常有一个参数来控制节点包含的数据实例最小到什么规模就不再分割。节点包含的数据实例太少会导致预测结果发生剧烈震荡。

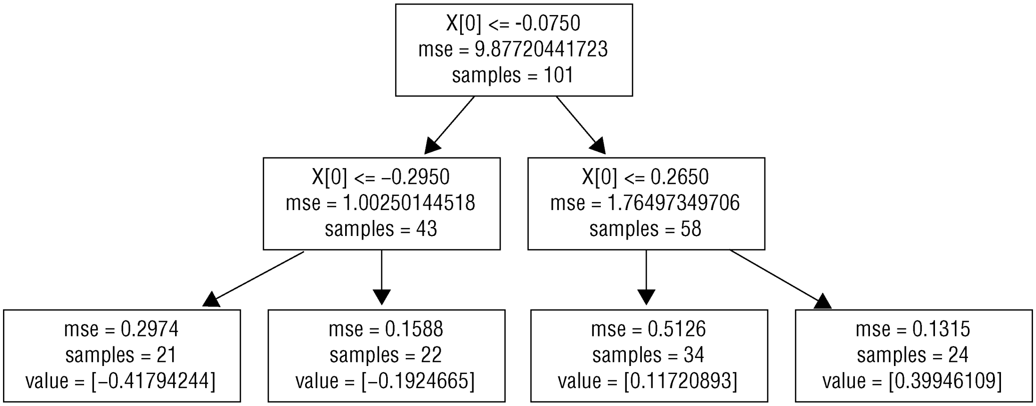


图 6-7 深度为 2 的决策树的框图

6.1.4 二元决策树的过拟合

上节介绍了如何训练任意深度的二元决策树。那么有没有可能过拟合一个二元决策树？本节介绍如何度量和控制二元决策树的过拟合。二元决策树的过拟合原因与第 4 章和第 5 章的有所不同。但是过拟合的表现以及如何度量过拟合过程还是比较相似的。二元决策树的参数（树的深度、最小叶节点规模等等）可以用来控制模型的复杂度，类似过程已经在第 4 章和第 5 章看到。

二元决策树过拟合的度量

图 6-8 展示了决策树的深度增加到 6 会发生什么。在图 6-8 中，很难看出真实值与预测值之间的差别。预测值几乎完全跟随每一个观察值的变化。这就开始暗示此模型已经过拟合了。数据产生方式表明最佳预测就是让预测值等于对应的属性值。添加到属性上的噪音是不可预测的，然而过拟合的预测结果是实际值加上噪声产生的偏差。合成数据的好处就是可以事先知道正确答案。

另外一个检查过拟合的方法是比较决策树中终止节点的数目与数据的规模。生成图 6-8 所示的预测曲线的决策树的深度是 6。这意味着它有 64 个终止节点（2^6）。数据集中共有 100 个数据点。这意味着大量的数据单独占据一个终止节点，因此它们的预测值与观察值完全匹配。这就不奇怪预测曲线完全跟随着噪声的“扭动”。

权衡二元决策树复杂度以获得最佳性能

在实际问题中，使用交叉验证（cross-validation）来控制过拟合。代码清单 6-3 展示了针对此问题使用不同深度的决策树运行 10 折交叉验证。代码显示了 2 层循环，外层循环定义了内层交叉验证的决策树深度，内层循环将数据分割为训练数据和测试数据后计算 10 轮

测试误差。不同深度的决策树对应的均方误差（MSE, mean squared error）如图 6-9 所示。

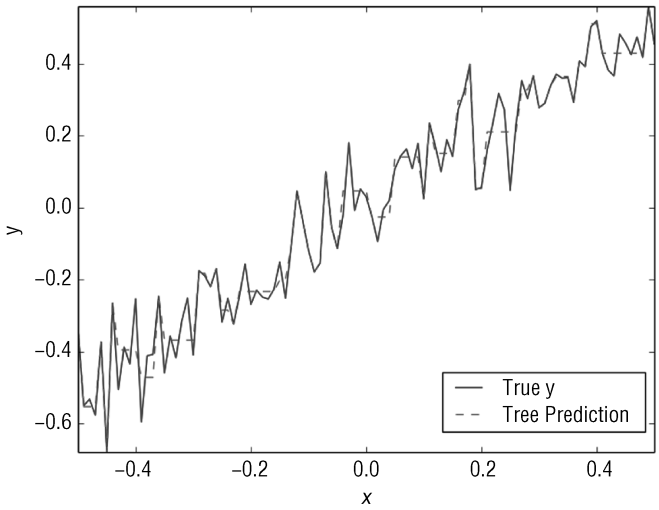


图 6-8 深度为 6 的决策树的预测曲线

代码清单 6-3 不同深度决策树的交叉验证 -simpleTreeCV.py

```
__author__ = 'mike-bowles'

import numpy
import matplotlib.pyplot as plot
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.externals.six import StringIO

#Build a simple data set with y = x + random
nPoints = 100

#x values for plotting
xPlot = [(float(i)/float(nPoints) - 0.5) for i in range(nPoints + 1)]

#x needs to be list of lists.
x = [[s] for s in xPlot]

#y (labels) has random noise added to x-value
#set seed
```

```
numpy.random.seed(1)
y = [s + numpy.random.normal(scale=0.1) for s in xPlot]

nrow = len(x)

#fit trees with several different values for depth and use
#x-validation to see which works best.

depthList = [1, 2, 3, 4, 5, 6, 7]
xvalMSE = []
nxval = 10

for iDepth in depthList:

    #build cross-validation loop to fit tree and evaluate on
    #out of sample data
    for ixval in range(nxval):

        #Define test and training index sets
        idxTest = [a for a in range(nrow) if a%nxval == ixval%nxval]
        idxTrain = [a for a in range(nrow) if a%nxval != ixval%nxval]

        #Define test and training attribute and label sets
        xTrain = [x[r] for r in idxTrain]
        xTest = [x[r] for r in idxTest]
        yTrain = [y[r] for r in idxTrain]
        yTest = [y[r] for r in idxTest]

        #train tree of appropriate depth and accumulate
        #out of sample (oos) errors
        treeModel = DecisionTreeRegressor(max_depth=iDepth)
        treeModel.fit(xTrain, yTrain)

        treePrediction = treeModel.predict(xTest)
        error = [yTest[r] - treePrediction[r] \
            for r in range(len(yTest))]

        #accumulate squared errors
        if ixval == 0:
```

```
            oosErrors = sum([e * e for e in error])
        else:
            #accumulate predictions
            oosErrors += sum([e * e for e in error])

    #average the squared errors and accumulate by tree depth

    mse = oosErrors/nrow
    xvalMSE.append(mse)

plot.plot(depthList, xvalMSE)
plot.axis('tight')
plot.xlabel('Tree Depth')
plot.ylabel('Mean Squared Error')
plot.show()
```

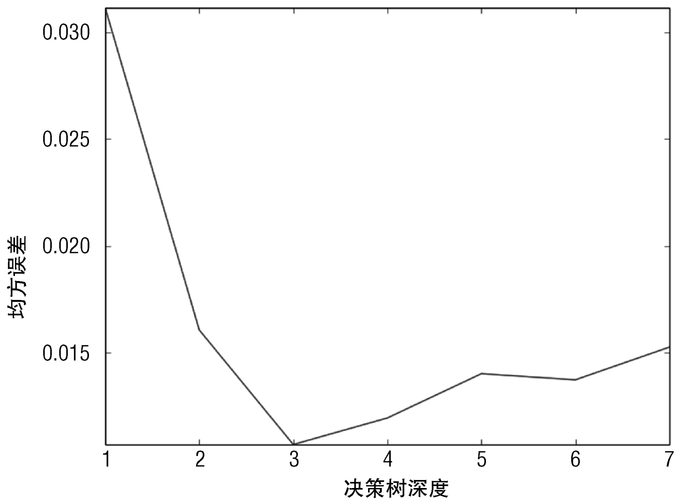


图 6-9 简单问题的测试数据均方误差与决策树深度的关系

决策树的深度控制二元决策树模型的复杂度。它的效果类似于第 4 章和第 5 章中惩罚回归模型的惩罚系数项。决策树深度的增加意味着在付出额外的复杂度的基础上，可以从数据中提取出更复杂的行为。图 6-9 说明决策树深度为 3 时，可以获得基于代码清单 6-2 生成的数据的最佳均方误差（MSE）。此决策树深度体现了重现属性与标签的内在关系和过拟合风险之间的最佳权衡。

回顾第 3 章，最佳模型的复杂度是数据集规模的函数。合成数据问题提供了观察这个

关系是如何起作用的机会。当数据点增加到 1 000 时，最佳模型复杂度和性能发生的变化如图 6-10 所示。

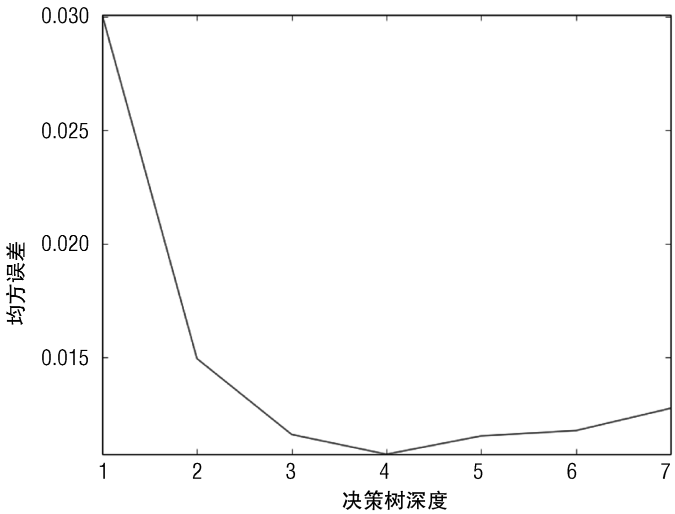


图 6-10　1 000 个数据点时，测试数据均方误差与决策树深度关系

可以修改代码清单 6-3 中的变量 nPoints 为 1000, 然后运行代码。增加数据时，会发生两件事情：第一件事是最佳决策树深度会从 3 增加到 4。增加的数据支持更复杂的模型。另外一件事是均方误差有轻微的下降。增加的决策树深度允许在逼近真实模型时提供更精细的“台阶”，面向真实的大规模数据场景也可以提供更好的保真度。

6.1.5 针对分类问题和类别特征所做的修改

为了提供关于决策树是如何训练的完整场景，还有一些细节问题需要讨论。一个问题就是：如何应用决策树解决分类问题？上述判断分割点的均方误差只对回归问题有意义。正如你在本书其他部分看到的，分类问题与回归问题有不同的评价标准。分类问题在判断分割点时可以使用多个评价标准来代替均方误差。一个是很熟悉的 misclassification error（误分类错误）。另外两个比较通用的是基尼不纯性度量（Gini impurity measure）和信息增益（information gain）。详细内容可以参考 http://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity。这两个度量指标与误分类错误有一些不同特性，但在概念上没有差别。

最后一个部分是当属性是类别属性而非数值属性时，如何训练决策树。决策树中的非终止节点提出一个 yes/no 的问题。对应数值属性，问题是判断属性是否小于某一值的这种形式。把一个类别属性（变量）分割成两个子集需要尝试所有分成 2 个子集的可能性。

假设一个类别属性包含 A、B、C 三类，可能的分割方式是：A 在一个子集，B、C 在另外一个子集，或者 B 在一个子集，A、C 在另外一个子集，诸如此类。在某些环境下，可以直接使用相关数学结果简化这个过程。

本节了提供二元决策树的背景知识，二元决策树本身就是一个很好的预测工具，值得深入研究。但是这里提出的目的是将其作为集成方法的背景。集成方法包含了大量的二元决策树。在当成千上万个决策树组合到一起时，使用单个决策树时出现的问题（如需要调整多个参数、结果的不稳定性、决策树深度加深导致的过拟合等）就会减弱。这也是提出集成方法的原因，集成方法更加鲁棒、易于训练、更加准确。下面讨论三个主流的集成方法。

6.2　自举集成：Bagging 算法

自举集成（bootstrap aggregation 即 Bagging 算法）是由 Leo Breiman 提出的。该方法是从选取一个基学习器开始的，本书使用二元决策树作为基学习器。随着对此方法介绍的深入，可以看到其他机器学习算法也可以作为基学习器。二元决策树是合乎逻辑地选择，因为它可以对具有复杂决策边界的问题建模，但二元决策树性能表现很不稳定。这种不稳定性可以通过组合多个基于决策树的模型来克服。

6.2.1　Bagging 算法是如何工作的

自举集成算法使用叫作自举（bootstrap）的取样方法。自举取样通常用来从一个中等规模的数据集中产生取样统计。一个（非参）自举取样是从数据集放回式地随机选择元素（也就是说，自举取样可能会重复取出原始数据中的同一行数据）。自举集成从训练数据集中获得一系列的自举样本，然后针对每一个自举样本训练一个基学习器。对于回归问题，结果为基学习器的均值。对于分类问题，结果是从不同类别所占的百分比引申出来的各种类别的概率或均值。代码清单 6-4 展示了对本章开始介绍的合成数据问题如何应用 Bagging 算法。

代码预留 30% 的数据作为测试数据，以代替交叉验证方法。参数 numTreesMax 决定集成方法包含的决策树的最大数目。代码建立模型是从第一个决策树开始，然后是前两个决策树、前三个决策树，以此类推，直到 numTreesMax 个决策树，可以看到预测的准确性与决策树数目之间的关系。代码将训练好的模型存入一个列表，并且存储了测试数据的预测值，这些预测值用于评估测试误差。代码画了两个图，一个展示了当集成方法增加决策树时，均方误差是如何变化的。另外一个图展示了第一个决策树的预测值、前 10 个决策树的平均预测值和前 20 个决策树的平均预测值的对比图。这个对比分析图与预测值曲线和实际标签值的对比图十分相似。

代码清单 6-4 自举集成算法 -simpleBagging.py

```
__author__ = 'mike-bowles'

import numpy
import matplotlib.pyplot as plot
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from math import floor
import random

#Build a simple data set with y = x + random
nPoints = 1000

#x values for plotting
xPlot = [(float(i)/float(nPoints) - 0.5) for i in range(nPoints + 1)]

#x needs to be list of lists.
x = [[s] for s in xPlot]

#y (labels) has random noise added to x-value
#set seed
random.seed(1)
y = [s + random.normal(scale=0.1) for s in xPlot]

#take fixed test set 30% of sample
nSample = int(nPoints * 0.30)
idxTest = random.sample(range(nPoints), nSample)
idxTest.sort()
idxTrain = [idx for idx in range(nPoints) if not(idx in idxTest)]

#Define test and training attribute and label sets
xTrain = [x[r] for r in idxTrain]
xTest = [x[r] for r in idxTest]
yTrain = [y[r] for r in idxTrain]
yTest = [y[r] for r in idxTest]

#train a series of models on random subsets of the training data
#collect the models in a list and check error of composite as list grows
```

```
#maximum number of models to generate
numTreesMax = 20

#tree depth - typically at the high end
treeDepth = 1

#initialize a list to hold models
modelList = []
predList = []

#number of samples to draw for stochastic bagging
nBagSamples = int(len(xTrain) * 0.5)

for iTrees in range(numTreesMax):
    idxBag = random.sample(range(len(xTrain)), nBagSamples)
    xTrainBag = [xTrain[i] for i in idxBag]
    yTrainBag = [yTrain[i] for i in idxBag]

    modelList.append(DecisionTreeRegressor(max_depth=treeDepth))
    modelList[-1].fit(xTrainBag, yTrainBag)

    #make prediction with latest model and add to list of predictions
    latestPrediction = modelList[-1].predict(xTest)
    predList.append(list(latestPrediction))

#build cumulative prediction from first "n" models
mse = []
allPredictions = []
for iModels in range(len(modelList)):

    #average first "iModels" of the predictions
    prediction = []
    for iPred in range(len(xTest)):
        prediction.append(sum([predList[i][iPred] \
            for i in range(iModels + 1)])/(iModels + 1))

    allPredictions.append(prediction)
    errors = [(yTest[i] - prediction[i]) for i in range(len(yTest))]
    mse.append(sum([e * e for e in errors]) / len(yTest))
```

```
nModels = [i + 1 for i in range(len(modelList))]

plot.plot(nModels,mse)
plot.axis('tight')
plot.xlabel('Number of Models in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()

plotList = [0, 9, 19]
for iPlot in plotList:
    plot.plot(xTest, allPredictions[iPlot])
plot.plot(xTest, yTest, linestyle="--")
plot.axis('tight')
plot.xlabel('x value')
plot.ylabel('Predictions')
plot.show()
```

图 6-11 展示了当决策树数目增加时均方误差是如何变化的。误差在 0.025 左右稳定下来。这个结果并不好。添加的噪声标准差为 0.1。一个预测算法的最佳均方误差应该是这个标准差的平方，也就是 0.01。本章前面的单个二进制决策树就已经接近 0.01 了。为什么复杂的算法性能反倒下降？

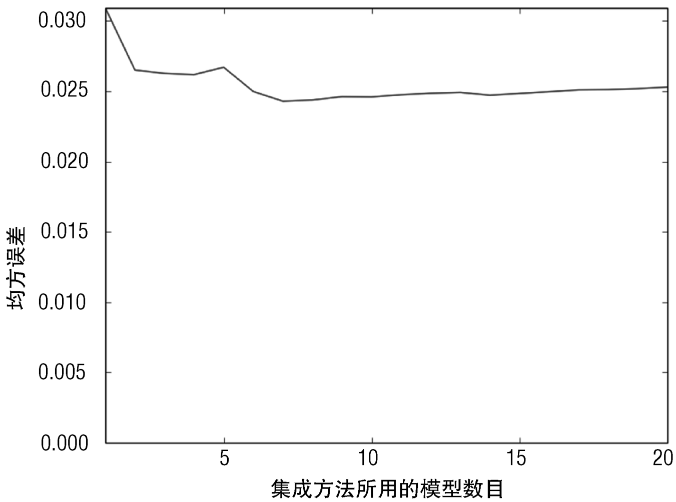


图 6-11 在 Bagging 集成方法中均方误差与决策树数目关系图

Bagging 的性能 - 偏差与方差（bias vs.variance）

仔细观察图 6-12 会对这个问题提供些启示，其中有一个重要的关键点需要明确指出，这个关键点与其他问题也有关系。图 6-12 展示了单个决策树的预测结果、10 个决策树的预测结果、20 个决策树的预测结果。单个决策树的预测值很容易看清楚，因为只有一个“台阶”。10 个决策树和 20 个决策树的预测实际上叠加了一系列稍有不同的决策树，因此它们实际上是在单个决策树那个单一台阶的附近增加了一系列更精细的“台阶”。多个决策树的台阶并不都在同一个点上，因为它们是基于不同的取样数据进行训练的，导致了分割点有一定的随机性。但随机性只会在分割点附近的小范围内带来轻微扰动。因此，结果看起来变化并不大，因为所有的决策树对于应该在哪里进行分割是有大概共识的。

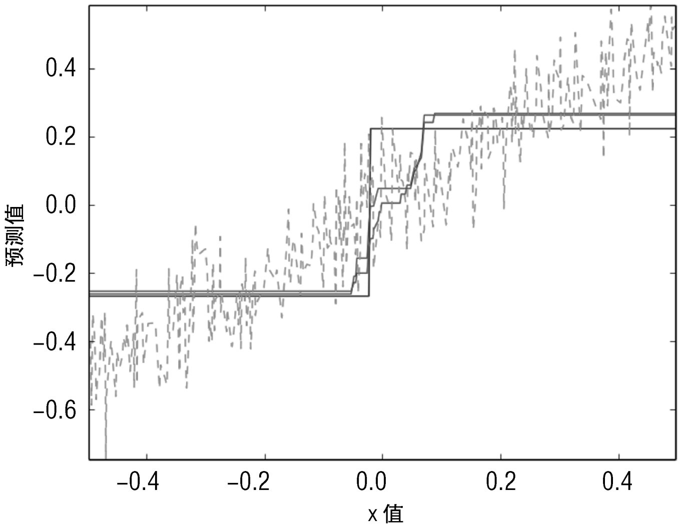


图 6-12 作为属性函数的实际标签值与预测值的关系图

这里存在两种误差：偏差和方差。考虑用一条直线来拟合一个摆动的曲线。增加数据规模可以减少噪声对拟合数据的影响，但是再多的数据也不可能让直线完全匹配一个摆动的曲线。当更多的数据点加入时，并不能减少的误差叫作偏差（bias errors)。用深度为 1 的决策树对合成数据拟合就会有偏差。所有的分割点都选在数据中心附近，因此对边缘数据的预测会影响模型的准确率。

深度为 1 的决策树的偏差来自于模型太简单了。Bagging 方法减少了模型之间的方差。但是对深度为 1 的决策树的偏差错误则是平均不掉的。克服这个问题的方法就是增加决策树的深度。

对应集成方法采用深度为 5 的决策树时，均方误差与决策树数目的关系图如图 6-13

所示。当采用深度为 5 的决策树时，均方误差稍微小于 0.01（可能是由于噪声数据的随机性），明显性能要好于深度为 1 的决策树。

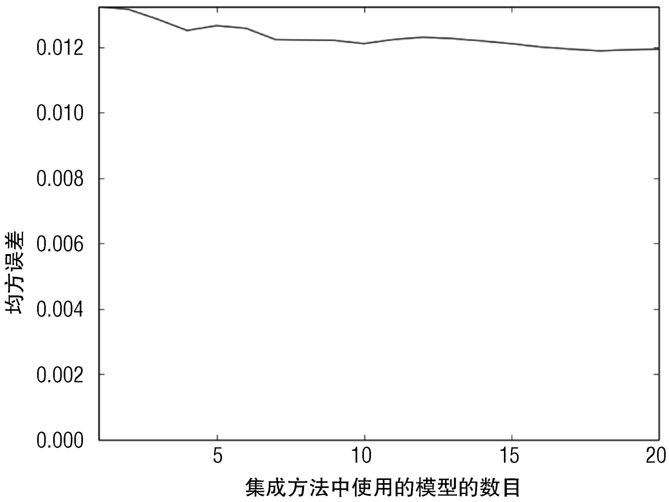


图 6-13 采用深度为 5 的决策树，均方误差与决策树数目之间的关系

使用 1 个决策树、10 个决策树、20 个决策树的预测结果如图 6-14 所示。单个决策树的预测结果明显比其他的突出，因为它有几个尖锐的突出，这导致了严重的误差。换句话说，单个决策树有高的方差。其他单个的决策树毫无疑问具有相似的性能。但是当它们被平均时（采用集成方法），方差减少了；基于 Bagging 算法的预测曲线更光滑，更接近于真实值。

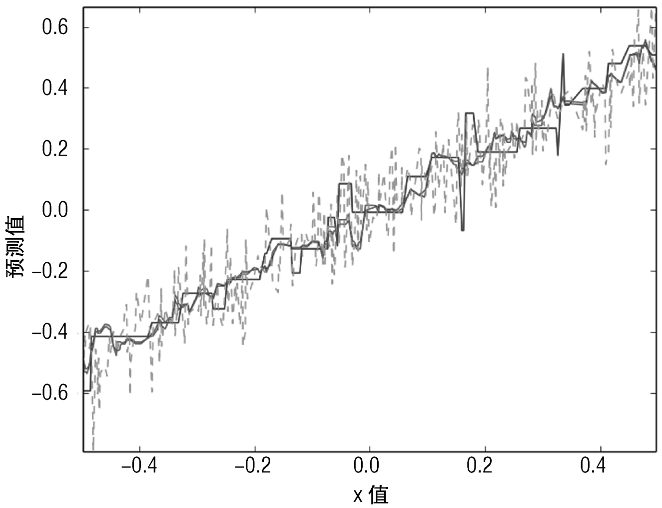


图 6-14 采用深度为 5 的决策树，预测值与真实值（标签）的关系

Bagging 算法如何解决多变量问题

代码清单 6-5 展示了 Bagging 算法如何解决预测红酒口感问题。红酒口感例子展示了与合成数据问题一致的处理原则，如图 6-15 ～图 6-17 所示。这些图是通过设置不同的参数，运行代码清单 6-5 获得的。

代码清单 6-5　用 Bagging 算法预测红酒口感 -wineBagging.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
import random
from math import sqrt
import matplotlib.pyplot as plot

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
```

```
        xList.append(floatRow)

nrows = len(xList)
ncols = len(xList[0])

#take fixed test set 30% of sample
random.seed(1)
nSample = int(nrows * 0.30)
idxTest = random.sample(range(nrows), nSample)
idxTest.sort()
idxTrain = [idx for idx in range(nrows) if not(idx in idxTest)]

#Define test and training attribute and label sets
xTrain = [xList[r] for r in idxTrain]
xTest = [xList[r] for r in idxTest]
yTrain = [labels[r] for r in idxTrain]
yTest = [labels[r] for r in idxTest]

#train a series of models on random subsets of the training data
#collect the models in a list and check error of composite as list grows

#maximum number of models to generate
numTreesMax = 30

#tree depth - typically at the high end
treeDepth = 1

#initialize a list to hold models
modelList = []
predList = []

#number of samples to draw for stochastic bagging
nBagSamples = int(len(xTrain) * 0.5)

for iTrees in range(numTreesMax):
    idxBag = []
    for i in range(nBagSamples):
        idxBag.append(random.choice(range(len(xTrain))))
    xTrainBag = [xTrain[i] for i in idxBag]
```

```
    yTrainBag = [yTrain[i] for i in idxBag]

    modelList.append(DecisionTreeRegressor(max_depth=treeDepth))
    modelList[-1].fit(xTrainBag, yTrainBag)

    #make prediction with latest model and add to list of predictions
    latestPrediction = modelList[-1].predict(xTest)
    predList.append(list(latestPrediction))


#build cumulative prediction from first "n" models
mse = []
allPredictions = []
for iModels in range(len(modelList)):

    #average first "iModels" of the predictions
    prediction = []
    for iPred in range(len(xTest)):
        prediction.append(sum([predList[i][iPred] \
            for i in range(iModels + 1)])/(iModels + 1))

    allPredictions.append(prediction)
    errors = [(yTest[i] - prediction[i]) for i in range(len(yTest))]
    mse.append(sum([e * e for e in errors]) / len(yTest))


nModels = [i + 1 for i in range(len(modelList))]
plot.plot(nModels,mse)
plot.axis('tight')
plot.xlabel('Number of Tree Models in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()

print('Minimum MSE')
print(min(mse))

#with treeDepth = 1
#Minimum MSE
```

```
#0.516236026081

#with treeDepth = 5
#Minimum MSE
#0.39815421341

#with treeDepth = 12 & numTreesMax = 100
#Minimum MSE
#0.350749027669
```

图 6-15 展示了在 Bagging 集成方法中包含更多决策树时，均方误差如何变化。基于树桩（stumps，深度为 1 的决策树）的集成方法相对于单个决策树在均方误差方面的改善可以忽略不计。与合成数据相比，红酒数据更加没有体现性能上的改善主要有以下几个原因：一方面红酒口感数据集边缘数据产生的误差更加显著，另外一方面变量（属性）之间相关性（相互作用）在红酒数据上更加突出。

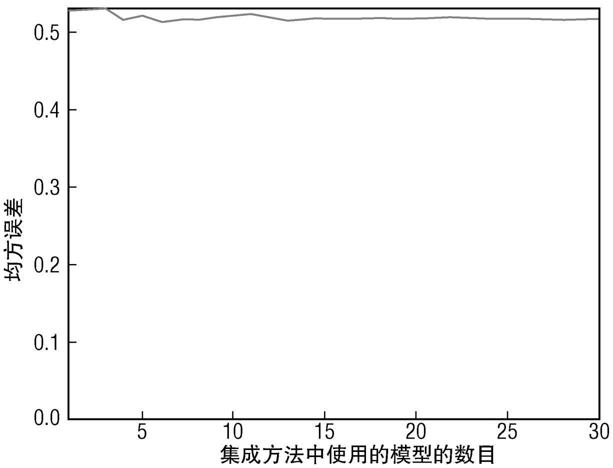


图 6-15 深度为 1 的决策树，Bagging 方法预测红酒口感

合成数据只有一个变量（属性），因此不存在属性之间的相互影响。红酒数据有多个属性，因此很可能属性的组合对预测的贡献要大于单独每个属性对预测的贡献和。“当你走路的时候被绊倒了，这个问题不大。如果你沿着悬崖走，问题也不严重。但是如果在沿着悬崖走的时候被绊倒了，这个问题可就严重了”（屋漏偏逢连夜雨）。上述两种可能性都要考虑。深度为 1 的决策树只能考虑单独的属性，没有考虑到属性之间相互影响的情况。

Bagging 算法为达到一定性能需要的决策树深度

图 6-16 展示了决策树深度为 5 时，均方误差与决策树数目的变化关系。随着更多的决策树加入，Bagging 集成方法的性能有明显改善。最终的性能要远远好与深度为 1 的决策树。这种改善暗示了当加入更多的决策树时，性能会进一步提高。

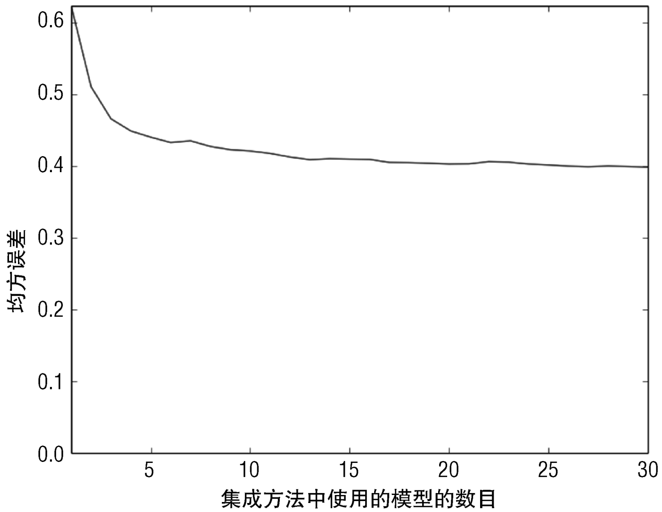


图 6-16 深度为 5 的决策树，Bagging 方法预测红酒口感

图 6-17 展示了决策树深度为 12 时，Bagging 集成方法的均方误差与决策树数目的关系。不仅可以采用更深的决策树，含有更多的决策树数目也可以进一步提升 Bagging 集成方法的性能。图 6-17 展示的是运行三次中最低的均方误差。

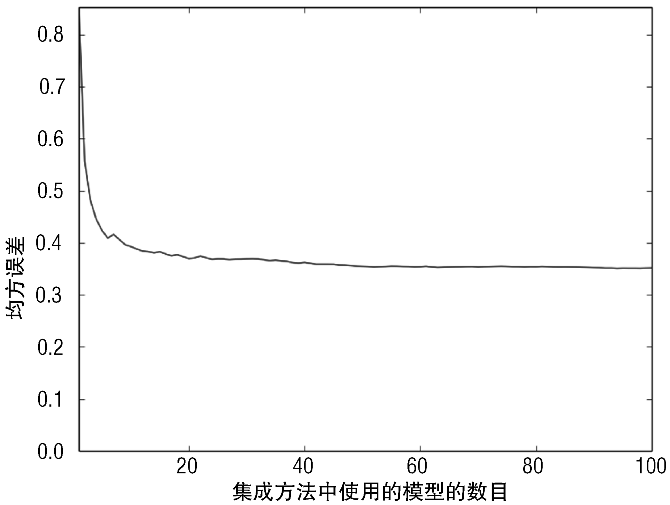


图 6-17 深度为 12 的决策树，Bagging 方法预测红酒口感

6.2.2 Bagging 算法小结

本节见证了集成方法的第一个例子。Bagging 方法展示的二级层次架构对集成方法来讲很普遍。准确地说 Bagging 是第二层次的算法：它定义了一系列的子问题，每个子问题由基学习器来解决，最终预测结果取各个基学习器预测的平均值。Bagging 集成方法的这些子问题是从原始训练数据中采用自举取样产生的。Bagging 方法可以减少单个二元决策树的方差。为了保证效果，Bagging 方法采用的决策树需要具有足够的深度。

Bagging 方法可以作为集成方法的入门介绍，因为它比较简单，易于理解，而且易于证明它可以减少方差的特性。下面介绍的算法是梯度提升法（gradient boosting）和随机森林（random forests）。它们采用不同的方法进行集成，并且显示了优于 Bagging 方法的特性。人们通常首先尝试梯度提升法或者随机森林，但是不常使用 Bagging。

6.3 梯度提升法（Gradient Boosting）

梯度提升法由斯坦福教授 Jerome Frideman 提出，他也提出坐标下降法来解决 ElasticNet 问题（见第 4 章和第 5 章）。梯度提升法是基于决策树的集成方法，在不同标签上训练决策树，然后将其组合起来。对于回归问题，目标是最小化均方误差，每个后续的决策树是在前面决策树遗留的错误上进行训练。关于此算法的来源可以查看本章的参考文献。了解梯度提升法是如何工作的最简单的方法就是直接查看代码实现。

6.3.1 梯度提升法的基本原理

代码清单 6-6 展示了针对本章的合成数据问题如何应用梯度提升法。代码的前面部分是生成合成数据集。

代码清单 6-6 简单问题使用梯度提升法 -simpleGBM.py

```
__author__ = 'mike-bowles'

import numpy
import matplotlib.pyplot as plot
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from math import floor
import random

#Build a simple data set with y = x + random
nPoints = 1000
```

```
#x values for plotting
xPlot = [(float(i)/float(nPoints) - 0.5) for i in range(nPoints + 1)]

#x needs to be list of lists.
x = [[s] for s in xPlot]

#y (labels) has random noise added to x-value
#set seed
numpy.random.seed(1)
y = [s + numpy.random.normal(scale=0.1) for s in xPlot]

#take fixed test set 30% of sample
nSample = int(nPoints * 0.30)
idxTest = random.sample(range(nPoints), nSample)
idxTest.sort()
idxTrain = [idx for idx in range(nPoints) if not(idx in idxTest)]

#Define test and training attribute and label sets
xTrain = [x[r] for r in idxTrain]
xTest = [x[r] for r in idxTest]
yTrain = [y[r] for r in idxTrain]
yTest = [y[r] for r in idxTest]

#train a series of models on random subsets of the training data
#collect the models in a list and check error of composite as list grows

#maximum number of models to generate
numTreesMax = 30

#tree depth - typically at the high end
treeDepth = 5

#initialize a list to hold models
modelList = []
predList = []
eps = 0.3
```

```
#initialize residuals to be the labels y
residuals = list(yTrain)

for iTrees in range(numTreesMax):

    modelList.append(DecisionTreeRegressor(max_depth=treeDepth))
    modelList[-1].fit(xTrain, residuals)

    #make prediction with latest model and add to list of predictions
    latestInSamplePrediction = modelList[-1].predict(xTrain)

    #use new predictions to update residuals
    residuals = [residuals[i] - eps * latestInSamplePrediction[i] \
        for i in range(len(residuals))]

    latestOutSamplePrediction = modelList[-1].predict(xTest)
    predList.append(list(latestOutSamplePrediction))

#build cumulative prediction from first "n" models
mse = []
allPredictions = []
for iModels in range(len(modelList)):

    #add the first "iModels" of the predictions and multiply by eps
    prediction = []
    for iPred in range(len(xTest)):
        prediction.append(sum([predList[i][iPred]
            for i in range(iModels + 1)]) * eps)

    allPredictions.append(prediction)
    errors = [(yTest[i] - prediction[i]) for i in range(len(yTest))]
    mse.append(sum([e * e for e in errors]) / len(yTest))


nModels = [i + 1 for i in range(len(modelList))]

plot.plot(nModels,mse)
plot.axis('tight')
```

```
plot.xlabel('Number of Models in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()

plotList = [0, 14, 29]
lineType = [':', '-.', '--']
plot.figure()
for i in range(len(plotList)):
    iPlot = plotList[i]
    textLegend = 'Prediction with ' + str(iPlot) + ' Trees'
    plot.plot(xTest, allPredictions[iPlot], label = textLegend,
        linestyle = lineType[i])
plot.plot(xTest, yTest, label='True y Value', alpha=0.25)
plot.legend(bbox_to_anchor=(1,0.3))
plot.axis('tight')
plot.xlabel('x value')
plot.ylabel('Predictions')
plot.show()
```

梯度提升法参数设置

第一个与前面例子的不同之处就是单个决策树深度的设置。梯度提升法与 Bagging 和随机森林的不同之处在于它在减少方差的同时，还可以减少偏差。梯度提升法的特性就是它在树桩（深度为 1 的决策树）决策树的情况下，也可以获得同更深的决策树一样低的均方误差值。对于梯度提升法，只有属性之间有强烈的相互影响的情况下，才需要考虑增加决策树的深度。随着决策树深度的增加，性能获得了改善实际上可以作为判断属性之间是否存在相互影响的方法。

另一个不同点就是变量 eps 的定义。这个变量在优化问题时用来控制步长。梯度提升法使用了梯度下降法，就像其他梯度下降算法一样，如果步长太大，优化过程就会发散而不是收敛。如果步长太小，则需要执行太多次迭代。本章后续会讨论如何调整步长，即 eps 值。

下一段不太熟悉的代码就是关于变量残差（residuals）的定义。术语残差通常用于表示预测误差（在这里是：观测值减去预测值）。梯度提升法会对标签的预测值进行一系列精确化。沿着梯度下降的方向，每走一步，残差都会重新计算。在开始阶段，梯度提升法将初始化预测值为空（null）或 0，因此残差等于观测值。

梯度提升法如何通过迭代获得预测模型

对 iTrees 的循环是以用属性值训练一个决策树开始的，但是用残差代替标签进行训练。只有第一轮是用原始的标签来训练数据的。后续的循环都是用训练产生的预测值，然后用残差减去（eps* 预测值）作为目标结果进行训练。如前文提到的，残差减去的相当于梯度下降的值。乘以一个步长控制参数 eps 就是为了保证迭代过程的收敛。代码使用固定的预留数据集（测试数据）来测量性能，并绘制了均方误差与决策树数目的关系图以及预测值与单一属性值的关系图。

6.3.2　获取梯度提升法的最佳性能

图 6-18 和图 6-19 展示了均方误差与决策树数目的关系图，预测值与属性值的关系图，决策树深度为 1，eps=0.1。图 6-18 展示了均方误差平滑下降，在训练了 30 个决策树的情况下，大概达到 0.014。均方误差曲线向下，说明随着决策树的增加，其值会持续下降。

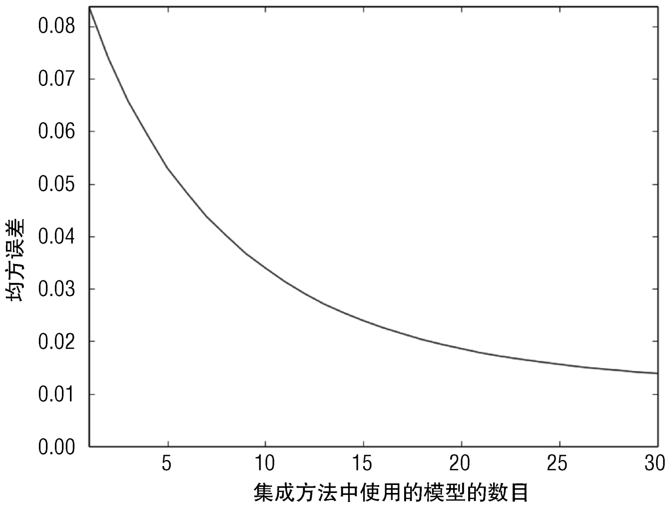


图 6-18　合成数据问题均方误差与决策树数目的关系（eps=0.1, 决策树深度为 1）

图 6-19 为三个梯度提升模型下，属性值与预测值之间的关系：只训练一个决策树、训练 15 个决策树、训练 30 个决策树。训练一个决策树的模型就像是我们在决策树介绍章节见到过的决策树模型的减弱版本。它实际上是一个深度为 1 的决策树，基于标签进行训练，然后再乘以 0.1（eps 的值）。当模型使用 10 个决策树时，有趣的现象出现了。模型实现了对正确答案（一个 45 度角的直线）很好的逼近。使用 10 个决策树的模型大概可以正确预测一半的边界，其右边、左边的预测值为常数。使用 30 个决策树的模型在每

个数据的边界都实现了很好的逼近。在这一点上与采用树桩（stumps）决策树的 Bagging 方法有很大区别。

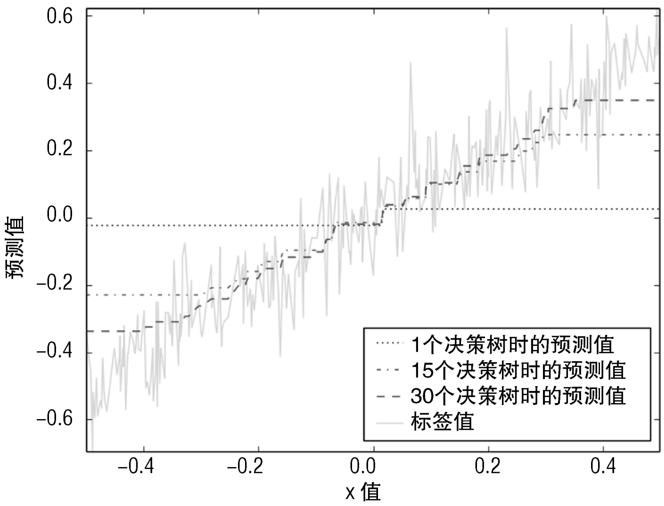


图 6-19 梯度提升法，预测值与属性值的关系（eps=0.1, 决策树深度为 1）

Bagging 方法不能改善偏差（bias error），这是由浅决策树固有特性决定的。梯度提升法以同样的方式开始，但是随着它开始减少过程中产生的错误，它开始更关注犯错误的区域。在会发生错误的地方设立分割点。这个过程使得不需要加深决策树的深度，就可以获得很好的逼近。

当控制训练的参数变化时会发生什么？当决策树的深度为 5 时发生的变化如图 6-20 和图 6-21 所示。当决策树的数目增加时，均方误差会平滑下降，如图 6-20 所示。当训练 30 个深度为 5 的决策树时，均方误差非常接近完美值（0.01）。图中没有显示训练时间。训练时，决策树的每一层大约花费相同的时间。在每一层，所有可能的分割点根据均方误差进行比较。因此深度为 5 的决策树所需时间是 5 个深度为 1 的决策树所需时间的 5 倍。通过比较可知，150 个深度为 1 的决策树的性能相当于 30 个深度为 5 的决策树。

决策树深度对梯度提升方法的影响如图 6-21 所示。即使只采用了单独一个决策树，其预测值在整个属性值区间也显示了一些变化。基于 15 个决策树的模型和 30 个决策树的模型在数据的边界仍然有一定的误差。

图 6-22 和图 6-23 展示了增加步长参数 eps 会发生什么。图 6-22 展示了步长（eps）太大会有什么特性。均方误差随着决策树数目的增加而急剧下降，然后又缓慢增加。其最小值是在图的左边，接近三分之一的地方。可以调整 eps 的值，使均方误差最小值在或者接近图的右侧，这样通常可以获得更佳的性能。

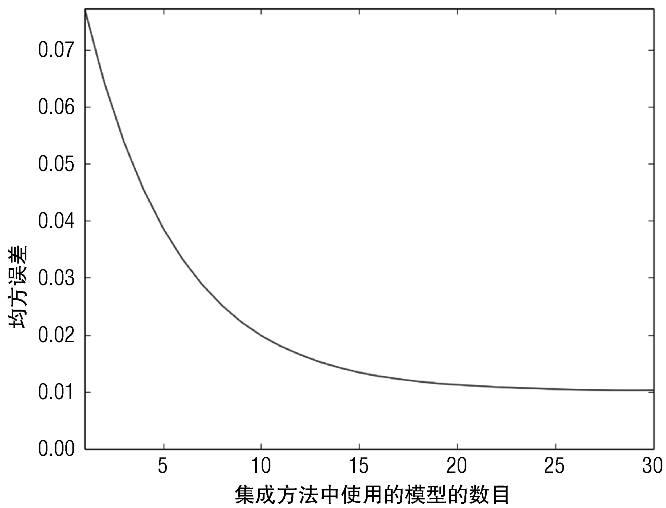


图 6-20 合成数据问题，均方误差与决策树数目的关系，eps=0.1, 决策树深度为 5

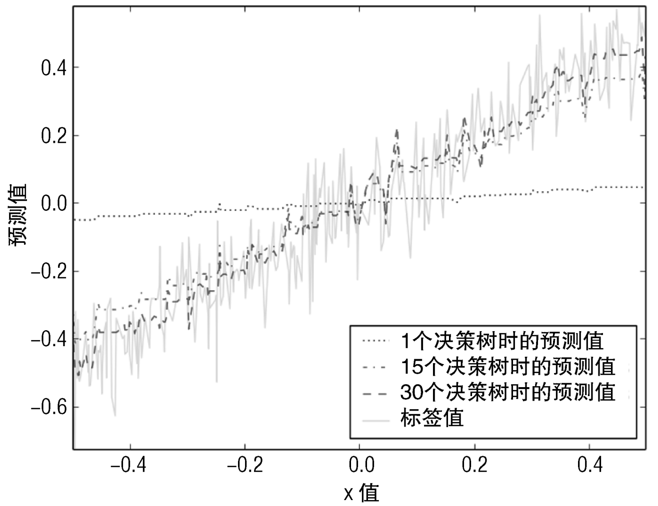


图 6-21 梯度提升法，预测值与属性值的关系，eps=0.1，决策树深度为 5

如图 6-23 所示，预测值作为属性值的函数 eps 为 0.3 的版本要比 eps 为 0.1 的两个版本（深度为 1 的决策树、深度为 5 的决策树）沿 45% 的直线显示了更多分散的突起。总的来看，深度为 1 的决策树显示了最佳的性能。这说明训练更多的决策树，会进一步改善模型在数据边界上的性能，从而获得梯度提升的最佳性能。

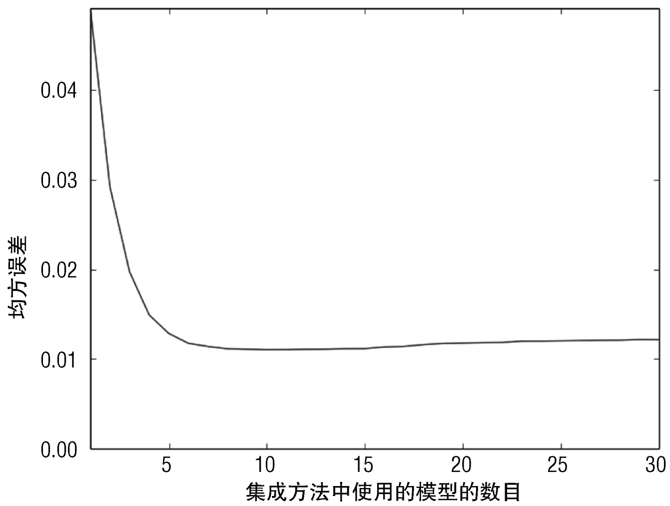


图 6-22 合成数据问题，均方误差与决策树数目的关系，eps=0.3, 决策树深度为 5

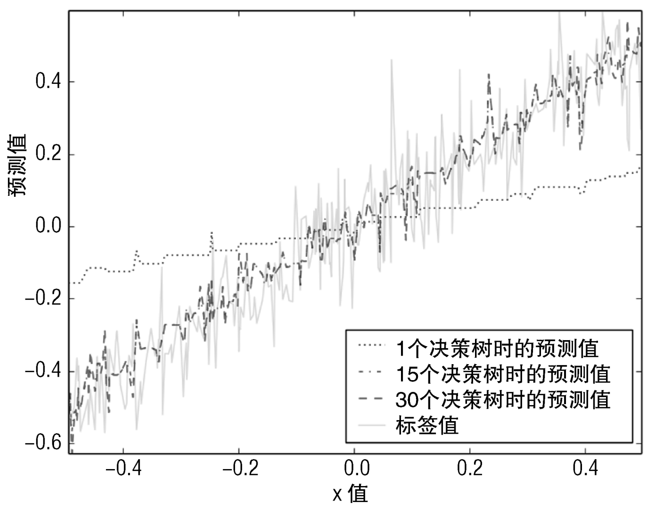


图 6-23 梯度提升法，预测值与属性值的关系，eps=0.3, 决策树深度为 5

6.3.3 针对多变量问题的梯度提升法

代码清单 6-7 展示了如何使用梯度提升法来预测红酒口感。除了使用红酒数据集作为输入外，此代码与用于合成数据的代码十分相似。

代码清单 6-7 使用梯度提升法预测红酒口感 -wineGBM.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
import random
from math import sqrt
import matplotlib.pyplot as plot

#read data into iterable
target_url = "http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

nrows = len(xList)
ncols = len(xList[0])
```

```
#take fixed test set 30% of sample
nSample = int(nrows * 0.30)
idxTest = random.sample(range(nrows), nSample)
idxTest.sort()
idxTrain = [idx for idx in range(nrows) if not(idx in idxTest)]

#Define test and training attribute and label sets
xTrain = [xList[r] for r in idxTrain]
xTest = [xList[r] for r in idxTest]
yTrain = [labels[r] for r in idxTrain]
yTest = [labels[r] for r in idxTest]

#train a series of models on random subsets of the training data
#collect the models in a list and check error of composite as list grows

#maximum number of models to generate
numTreesMax = 30

#tree depth - typically at the high end
treeDepth = 5

#initialize a list to hold models
modelList = []
predList = []
eps = 0.1

#initialize residuals to be the labels y
residuals = list(yTrain)

for iTrees in range(numTreesMax):

    modelList.append(DecisionTreeRegressor(max_depth=treeDepth))
    modelList[-1].fit(xTrain, residuals)

    #make prediction with latest model and add to list of predictions
    latestInSamplePrediction = modelList[-1].predict(xTrain)

    #use new predictions to update residuals
    residuals = [residuals[i] - eps * latestInSamplePrediction[i] \
```

```
            for i in range(len(residuals))]

        latestOutSamplePrediction = modelList[-1].predict(xTest)
        predList.append(list(latestOutSamplePrediction))


#build cumulative prediction from first "n" models
mse = []
allPredictions = []
for iModels in range(len(modelList)):

    #add the first "iModels" of the predictions and multiply by eps
    prediction = []
    for iPred in range(len(xTest)):
        prediction.append(sum([predList[i][iPred]
            for i in range(iModels + 1)]) * eps)

    allPredictions.append(prediction)
    errors = [(yTest[i] - prediction[i]) for i in range(len(yTest))]
    mse.append(sum([e * e for e in errors]) / len(yTest))


nModels = [i + 1 for i in range(len(modelList))]

plot.plot(nModels,mse)
plot.axis('tight')
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()

print('Minimum MSE')
print(min(mse))

#printed output
#Minimum MSE
#0.405031864814
```

选择的参数为：30 个深度为 5 决策树，eps=0.1。这个参数集产生的均方误差大致为 0.4。

对于同样的问题，此方法大概比 Bagging 的方法差 10%。可以尝试调整：决策树的数目、eps 步长和决策树的深度来看看是否可以获得更佳的性能。

均方误差与决策树数目的关系曲线在右边相当平坦，如图 6-24 所示。通过增加决策树的数目还是可能获得性能上的改善的。另外一个可能“压榨”性能的方法就是微调步长或决策树的深度。

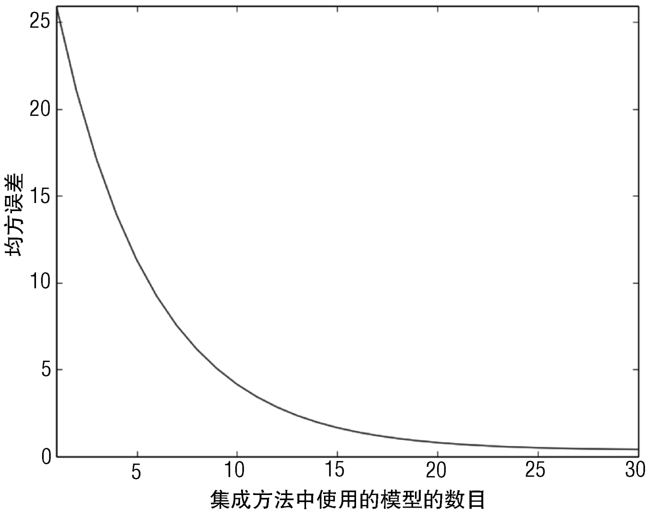


图 6-24 红酒口感问题，梯度提升法，均方误差与决策树数目的关系

6.3.4 梯度提升方法的小结

本节介绍了如何使用梯度提升方法、如何调整参数获得最佳性能，以及步长、决策树深度、决策树数目对性能的影响。还介绍了梯度提升法是如何避免偏差的，这个是 Bagging 方法采用浅决策树时会遇到的问题。Bagging 和梯度提升法在工作原理上的根本差异在于梯度提升法持续监测自己的累积误差，然后使用残差进行后续训练。这种根本差异也解释了为什么当问题属性之间存在强的相互依赖、相互作用时，梯度提升法只需要调整决策树的深度。

6.4 随机森林

随机森林算法由伯克利教授 Leo Breiman 和 Adele Cutler 联合提出。随机森林在数据集的子集上训练出一系列的模型。这些子集是从全训练数据集中随机抽取的。一种抽取方法就是对数据行的随机放回取样，同 Brieman 的自举集成方法一样。另一种方法是每个

决策树的训练数据集只是所有属性随机抽取的一个子集，而不是全部的属性。代码清单 6-8 用 Python 的 DecisionTreeRegression 来近似随机森林。

代码清单 6-8 随机选择属性的 bagging 方法 - wineRF.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
import random
from math import sqrt
import matplotlib.pyplot as plot

#read data into iterable
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)
```

```
nrows = len(xList)
ncols = len(xList[0])

#take fixed test set 30% of sample
random.seed(1) #set seed so results are the same each run
nSample = int(nrows * 0.30)
idxTest = random.sample(range(nrows), nSample)
idxTest.sort()
idxTrain = [idx for idx in range(nrows) if not(idx in idxTest)]

#Define test and training attribute and label sets
xTrain = [xList[r] for r in idxTrain]
xTest = [xList[r] for r in idxTest]
yTrain = [labels[r] for r in idxTrain]
yTest = [labels[r] for r in idxTest]

#train a series of models on random subsets of the training data
#collect the models in a list and check error of composite as list grows

#maximum number of models to generate
numTreesMax = 30

#tree depth - typically at the high end
treeDepth = 12

#pick how many attributes will be used in each model.
# authors recommend 1/3 for regression problem
nAttr = 4

#initialize a list to hold models
modelList = []
indexList = []
predList = []
nTrainRows = len(yTrain)

for iTrees in range(numTreesMax):
```

```
    modelList.append(DecisionTreeRegressor(max_depth=treeDepth))
    #take random sample of attributes
    idxAttr = random.sample(range(ncols), nAttr)
    idxAttr.sort()
    indexList.append(idxAttr)

    #take a random sample of training rows
    idxRows = []
    for i in range(int(0.5 * nTrainRows)):
        idxRows.append(random.choice(range(len(xTrain))))
    idxRows.sort()

    #build training set
    xRfTrain = []
    yRfTrain = []

    for i in range(len(idxRows)):
        temp = [xTrain[idxRows[i]][j] for j in idxAttr]
        xRfTrain.append(temp)
        yRfTrain.append(yTrain[idxRows[i]])

    modelList[-1].fit(xRfTrain, yRfTrain)

    #restrict xTest to attributes selected for training
    xRfTest = []
    for xx in xTest:
        temp = [xx[i] for i in idxAttr]
        xRfTest.append(temp)

    latestOutSamplePrediction = modelList[-1].predict(xRfTest)
    predList.append(list(latestOutSamplePrediction))


#build cumulative prediction from first "n" models
mse = []
allPredictions = []
for iModels in range(len(modelList)):
```

```
    #add the first "iModels" of the predictions and multiply by eps
    prediction = []
    for iPred in range(len(xTest)):
        prediction.append(sum([predList[i][iPred]
            for i in range(iModels + 1)]) / (iModels + 1))

    allPredictions.append(prediction)
    errors = [(yTest[i] - prediction[i]) for i in range(len(yTest))]
    mse.append(sum([e * e for e in errors]) / len(yTest))


nModels = [i + 1 for i in range(len(modelList))]

plot.plot(nModels,mse)
plot.axis('tight')
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()

print('Minimum MSE')
print(min(mse))

#printed output

#Depth 1
#Minimum MSE
#0.52666715461

#Depth 5
#Minimum MSE
#0.426116327584

#Depth 12
#Minimum MSE
#0.38508387863
```

6.4.1 随机森林：Bagging 加上随机选择的属性子集

代码清单 6-8 是对红酒口感数据集的训练，用来说明 Bagging 和梯度提升算法的单一属性的例子是不能用于随机森林的。那个例子只有一个属性。对于单独一个属性进行随机选取是没有意义的。代码清单 6-8 看起来很像 Bagging 的代码。两者之间唯一的差别就是在 iTrees 循环之前，指定了一个变量 nAttr。随机抽取属性时，需要知道要选取多少个属性。算法的提出者建议对于回归问题，选择全部属性的三分之一（对于分类问题，选择全部属性数目的平方根）。在 iTrees 循环内部，有对属性矩阵行的取样（这与 Bagging 相同）。还有一个对属性矩阵的列的随机不放回取样（如果是 numpy 的数组形式，就是行和列）。然后训练决策树，对测试数据进行预测。

代码清单 6-8 中的实现与随机森林算法还有区别。代码清单 6-8 取属性的一个随机子集，然后基于此子集训练决策树。Breiman 的原始版本对决策树的每个节点都用不同的属性随机子集进行训练。为了实现 Breiman 的原始版本，需要访问决策树生成算法的内部。这个例子只是给出了这个算法应该如何使用的大概思路，而且有人认为对决策树的每个节点都随机选择属性的意义并不大。

6.4.2 随机森林的性能

图 6-25 ～图 6-27 为增加属性的随机选择后，对均方误差与决策树数目的关系曲线的影响。图 6-25 为决策树深度为 1 时的结果。此图与 Bagging 方法十分相似，说明性能没有太大提高。深度为 1 的决策树主要导致了偏差，而不是方差。偏差是不能被平均的。

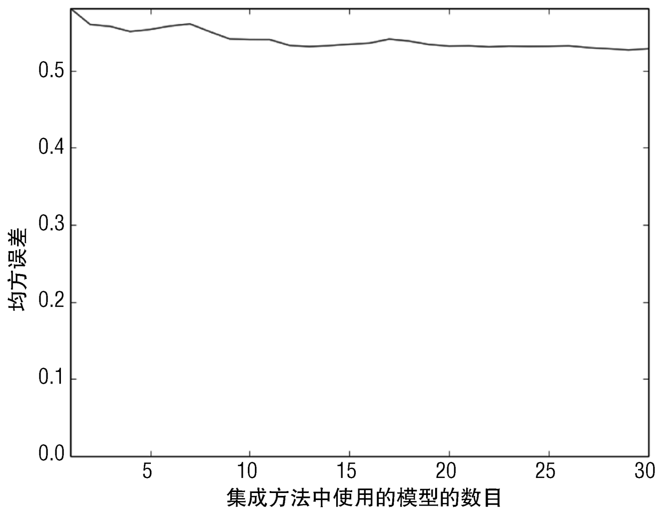


图 6-25 Bagging+ 属性随机选择，均方误差与决策树数目的关系，决策树深度为 1

图 6-26 为决策树深度为 5 时的均方误差曲线。通过 Bagging 方法减少了方差，再加上属性随机选择方法开始呈现了一定性能上的改善。这种组合可以达到与其他方法相似的性能。

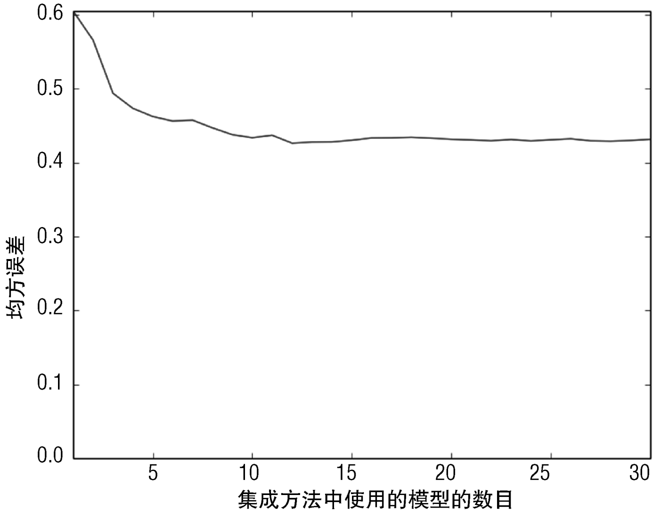


图 6-26 Bagging+ 属性随机选择，均方误差与决策树数目的关系，决策树深度为 5

图 6-27 说明当采用深度为 12 的决策树的时候，还可以更进一步提高性能。

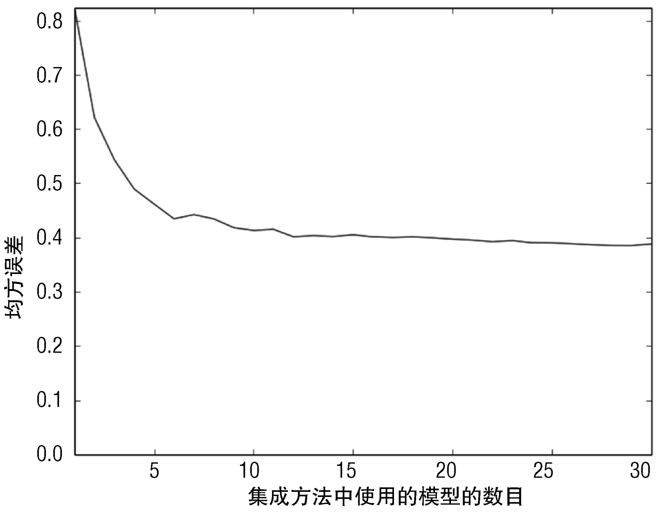


图 6-27 Bagging+ 属性随机选择，均方误差与决策树数目的关系，决策树深度为 12

6.4.3 随机森林小结

随机森林是两个方法的结合，包括 Bagging 方法和属性随机选择方法。属性随机选择

方法实际上是对二元决策树基学习器的修正。这些差异看起来不是本质上的，但是这些给予了随机森林与 Bagging 和梯度提升法不同的性能特性。有研究结果建议随机森林更适用于广泛稀疏的属性空间，如文本挖掘问题。与梯度提升法相比，随机森林更易于并行化，因为每个基学习器都可以单独训练。然而梯度提升法不行，因为每个基学习器都依赖于前一个基学习器的结果。

这些差异说明除了梯度提升法以外，还可以尝试随机森林以获取最佳性能。

小结

本章介绍了集成方法的背景知识。集成方法由两层算法组成。集成方法训练成百上千个叫作基学习器的低层算法，上层的算法控制基学习器的训练，使这些基学习器近乎相互独立，这样将这些基学习器组合起来就可以减少组合后的误差方差。Bagging 方法对训练数据集进行自举抽样（bootstrap sample, 在一个原始样本中进行有放回的重复抽样），然后基于这些抽样训练基学习器。梯度提升方法在每一步对输入数据进行抽样，然后基于这一样本训练基学习器。梯度提升法训练每个基学习器的目标是前期所有基学习器的累积误差。随机森林是将 Bagging 作为高层算法，将修改版的二元决策树作为基学习器。随机森林的基学习器是二元决策树，分割点的选择是基于所有属性的一个随机取样，而不是考虑所有属性。Python 的梯度提升工具包允许将随机森林作为梯度提升法的基学习器。第 7 章会详细介绍这一点。

本章展示了集成方法每个上层算法的代码和随机森林基学习器的“摹本”，目的是让读者理解每个算法的工作机制。这种方式有助于更好地理解 Python 对应算法包的选项、输入变量、归一化初始值等。下一章将介绍如何使用 Python 工具包解决惩罚线性回归章节中遇到的问题。

参考文献

1. Panda Biswanath , Joshua S. Herbach , Sugato Basu , and Roberto J. Bayardo .(2009). PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce. Proceedings of the 35th International Conference on Very Large Data Bases. Retrieved from http://research.google.com/pubs/pub36296.html .

2. Leo Breiman . (September, 1994). Bagging Predictors. Technical Report No. 421. Department of Statistics, UC Berkeley. Retrieved from http:// statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf .

3. Leo Breiman . (2001). Random forests . Machine Learning g , 45 : 5 – 32 . Retrieved

from http://oz.berkeley.edu/~breiman/randomforest2001.pdf .

4. J.H. Friedman . (2001). Greedy Function Approximation: A Gradient Boosting Machine . Annals of Statistics,29(5): 1189–1232.Retrieved from http://statweb. stanford.edu/~jhf/ftp/trebst.pdf.

5. J.H. Friedman . (2002). Stochastic Gradient Boosting . Computational Statistics and Data Analysis , 38 (4): 367– 378 . Retrieved from http://statweb.stanford.edu/~jhf/ftp/stobst.pdf .

第 7 章

用 Python 构建集成模型

本章使用 Python 工具包利用第 6 章介绍的集成方法构建预测模型。解决的问题在第 2 章中已介绍过。第 5 章介绍了如何使用惩罚线性回归来构建预测模型解决上述问题。本章使用集成方法来解决同样的问题。这样可以对集成方法与惩罚线性回归做多方位的比较：算法、Python 工具包的易用性、所能达到的准确性、训练所需的时间等等。本章的最后是各种算法的对比总结。

7.1 用 Python 集成方法工具包解决回归问题

下面介绍如何使用构建集成方法模型的 Python 工具包，将实际应用第 6 章的知识。本章使用第 6 章介绍的方法来解决第 2 章提出的问题，并与第 5 章的惩罚线性回归方法进行对比分析。通过解决同样的问题，可以对算法进行多维度的比较，包括性能、训练时间、易用性等。本章还将介绍这些 Python 工具包的使用。第 6 章的知识将有助于理解为什么 Python 工具包如此设计，以及如何充分利用这些方法的特性。本节将解决不同类型的问题，从回归问题开始。

7.1.1 构建随机森林模型来预测红酒口感

通过红酒口感数据集可以利用红酒中的化学成分来预测红酒口感的分数。众所周知，此问题是一个回归问题，因为预测值是实数的形式。Python scikit-learn 算法包集成方法模块中有随机森林和梯度提升法，这两个算法都可以用来解决回归问题。首先解释初始化 RandomForestRegressor 类成员所需的参数，然后使用 RandomForestRegressor 类来训练一个随机森林模型，将此模型应用于红酒数据集，测试模型的性能。

构建 RandomForestRegressor 对象

sklearn.ensemble.RandomForestRegressor 的类构造函数如下。

```
    sklearn.ensemble.RandomForestRegressor(n_estimators=10, criterion='mse',
max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features= 'auto',
```

```
max_leaf_nodes=None, bootstrap=True, oob_score=False, n_jobs=1, random_
state=None, verbose=0, min_density=None, compute_importances=None)
```

下面描述来自 sklearn 官方文档，但只选取了最可能需要修改的参数。对于这些参数，下面介绍如何选择其他值来代替缺省值。其他参数的介绍可参考 sklearn 官方文档。

◆ n_estimators

整型，可选（缺省值为 10）。

此参数指定集成方法中决策树的数目。通常缺省值就可以工作得很好。如果想获得最佳的性能，就需要多于 10 个决策树。可以通过做实验尝试确定最佳的决策树数目。正如全书始终强调的，合适的模型复杂度（决策树的深度和决策树的数目）取决于问题的复杂度和可获得的数据的规模。比较好的尝试是 100 ～ 500。

◆ max_depth

整型或者 none, 可选（缺省值为 None)。

如果这个参数设置为 None，决策树就会持续增长，直到叶子节点为空或者所含数据实例小于 min_samples_split。除了指定决策树的深度，可以用参数 max_leaf_nodes 来指定决策树的叶子节点数。如果指定了 max_leaf_nodes，max_depth 参数就会被忽略。不设置 max_depth，让决策树自由生长，形成一个满度的决策树可能可以获得性能上的好处。当然与之相伴的代价就是训练时间。在模型的训练过程中，可能需要尝试不同深度的决策树。

◆ min_sample_split

整型，可选（缺省值为 2）。

当节点含有的数据实例少于 min_sample_split 时，此节点不再分割。对含有较少实例的节点进行分割是过拟合错误的源头。

◆ min_samples_leaf

整型，可选（缺省值为 1）。

如果分割导致节点拥有的数据实例少于 min_sample_leaf，分割就不会进行。这个参数的缺省值实际上导致此参数被忽略。通常这是可行的，特别是对数据集进行头几次试运行时。可以用各种方法为这个参数选择一个更有意义的值。一个方法是参数选取为叶子节点含有实例数的平均值，这样如果叶节点含有多于 1 个的数据实例，就可以获得更低的均方差。另一种方法是将此参数看作控制决策树深度的替代方法。

◆ max_features

整型、浮点型或字符串型，可选（缺省值为 None)。

当查找最佳分割点时，需要考虑多少个属性是由 max_features 参数和问题中一共有多少个属性共同决定的。假设问题数据集中共有 nFeatures 个属性。则：

- 如果 max_features 是整型，则在每次分割时考虑 max_features 个属性。注：如果 max_features > nFeatures，则抛出错误。
- 如果 max_features 是浮点型，max_features 表示需考虑的属性占全部属性的百分比，即 int(max_features*nFeatures)。
- 可选择的字符串值如下：

```
auto max_features=nFeatures
sqrt max_features=sqrt(nFeatures)
log2 max_features=log2(nFeatures)
```

- 如果 max_features=None, 则 max_features=nFeatures

Brieman 和 Cutler 建议对回归问题使用 sqrt(nFeatues) 个属性。模型通常对 max_features 不是很敏感，但是这个参数还是有一些影响，因此可以根据需要尝试一些不同的值。

◆ random_state

整型，RandomState[①] 实例，或者 None (缺省值为 None)。

- 如果类型是整型，则此整数作为随机数生成器的种子。
- 如果是 RandomState 的一个实例，则此实例用来作为随机数生成器。
- 如果是 None, 则随机数生成器是 numpy.random 用的 RandomState 的一个实例。

RandomForestRegressor 类有几个属性，包括用来构成集成方法的决策树。RandomForestRegressor 类有用训练好的决策树进行预测的方法，因此通常不需要直接访问这些属性。但是可能需要访问变量 importances。下面是对此变量的描述。

◆ feature_importances

这是一个数组，数组的长度等于问题的属性数（也就是 nFeatures)。数组中的值是正的浮点数，表明对应的属性对预测结果的贡献重要性。属性的重要性由 Brieman 在最初的随机森林论文中所提的一个方法来确定。基本思想是，每次选中一个属性，然后对属性的值进行随机置换，记录下预测准确性的变化，预测的准确性越高，此属性也越重要。

下面是对类的方法的描述。

◆ fit(XTrain, yTrain, sample_weight=None)

XTrain 是属性值的数组（训练数据),它有 nInstances 行和 nFeatures 列。yTrain 是目标（标签）值的数组。y 同样有 nInstances 行。在本章的例子中可以看到 yTrain 只有一列，但是此方法可以应用于具有不同目标的模型。因此 y 可以有 nTargets 列，每列对应一个结果（目标，标签）集合。Sample_weight 用来对训练数据集中的每个实例分配不同的权重，

① RandomState : Python numpy 的 Mersenne Twister 伪随机数生成器。

它有两种形式：缺省值是 None，意味着所有输入实例具有相同的权重；如果对每个实例分配不同的权重，sample_weight 就是一个数组，具有 nInstances 行和 1 列。

◆ predict(XTest)

XTest 是属性值的数组（测试数据），基于这些属性值进行预测。此数组是 predict() 方法的输入，此数组的列数与输入 fit() 的数组的列数相同，但是可能具有不同的行数，也可能只有一行。Predict() 的输出形式与用于训练的目标数组 y 相同。

用 RandomForestRegressor 对红酒口感问题建模

代码清单 7-1 展示了如何用 sklearn 的随机森林算法构建集成方法模型来预测红酒口感。

代码首先从 UCI 数据仓库中读取红酒数据集；进行预处理获得属性、标签、属性名存入列表；将列表转换为 numpy 数组形式，此形式是 RandomForestRegressor 要求的。将这些输入对象转换为 numpy 数组形式的一个额外的好处是可以使用 sklearn 的 train_test_split 构建训练和测试集。代码将 random_state 设置为一个特殊的整数值，而不是让随机数生成器自己选择一个不可重复的内部值。这样重复运行代码时，可以获得同样的结果。设置 random_state 为固定的值，在开发阶段也为模型的调整提供了便利，否则随机性会掩盖所做的改变。在真实的模型训练阶段，可将 random_state 设为缺省值 None。固定 random_state 值就固定了测试集，这样重复的参数调整和训练最终是对测试数据集的过度训练。

代码的下一步是定义集成方法的不同规模（不同的决策树数目），由此产生性能曲线，展示当集成方法中决策树的数目发生变化时，性能是如何变化的。为了让性能曲线图足够精确，在代码清单 7-1 中大约选择 45 个不同的决策树规模来运行。选取这么多点运行是有意义的，可以看到决策树的数目与误差清晰的关系曲线，但是当在脑海中已经有了大致的印象后，就不需要跑这么多点了。在开发的早期阶段，可以尝试跑 2、3 个不同的决策树规模，然后确定一个比较好的决策树规模，在以后的绝大多数时间里都可以按这个规模进行实验。

绝大多数训练过程的参数都在 RandomForestRegressor 的构造函数中设置。在这个例子中构造函数的调用十分简单。唯一一个没有使用缺省值的参数就是 max_features。此参数的缺省值（None）会导致在决策树的每个节点都要考虑全部的属性。这意味着实际上实现的是 Bagging 方法，因为这里没有随机选择属性的过程。

初始化 RandomForestRegressor 对象后，下一步就是调用 fit() 方法，训练数据集作为其输入参数。最后是调用 predict() 方法进行预测，其输入是测试数据集的属性，并将预测值与测试数据集中的标签比较。代码用 sklearn.metrics 的 mean_squared_error 函数来计算

预测均方误差，并将其保存在一个列表中，结果如图 7-1 所示。

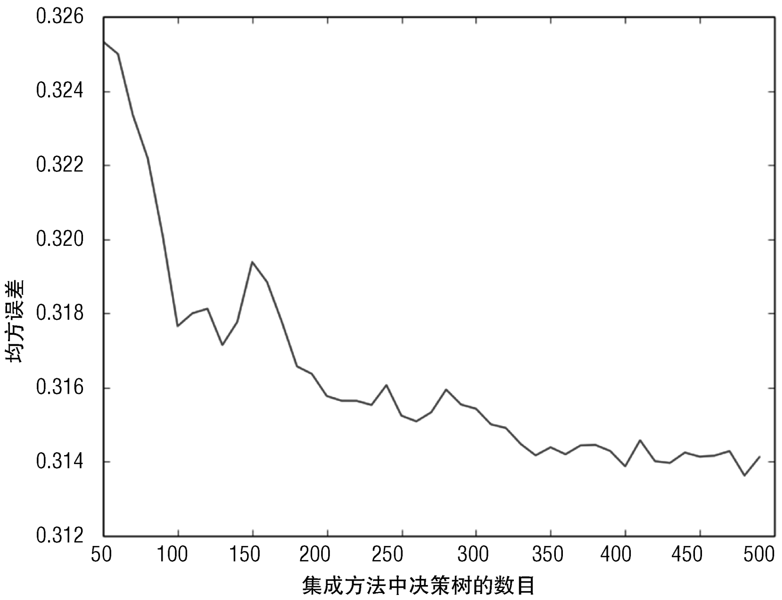


图 7-1　利用随机森林对红酒口感进行预测：均方误差与集成方法规模的关系

最后一个均方误差值打印在代码清单 7-1 的最后。最后一个均方误差值只是作为均方误差的代表被打印出来，并不代表最小值。随机森林产生近乎独立的预测，然后取它们的平均值。因为是取平均值，增加更多的决策树不会导致过拟合，因此图 7-1 曲线上的最小值是统计上的波动造成的，不是可重复的最小值。

代码清单 7-1　用 RandomForestRegressor 构建回归模型 – wineRF.py

```
import urllib2
import numpy
from sklearn.cross_validation import train_test_split
from sklearn import ensemble
from sklearn.metrics import mean_squared_error
import pylab as plot


# Read wine quality data from UCI website
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
databases/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)
```

```
xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

nrows = len(xList)
ncols = len(xList[0])

X = numpy.array(xList)
y = numpy.array(labels)
wineNames = numpy.array(names)

#take fixed holdout set 30% of data rows
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.30,
  random_state=531)

#train Random Forest at a range of ensemble sizes in order to
#see how the mse changes
mseOos = []
nTreeList = range(50, 500, 10)
for iTrees in nTreeList:
    depth = None
    maxFeat = 4 #try tweaking
```

```
    wineRFModel = ensemble.RandomForestRegressor(n_estimators=iTrees,
        max_depth=depth, max_features=maxFeat,
        oob_score=False, random_state=531)

    wineRFModel.fit(xTrain,yTrain)

    #Accumulate mse on test set
    prediction = wineRFModel.predict(xTest)
    mseOos.append(mean_squared_error(yTest, prediction))


print("MSE" )
print(mseOos[-1])


#plot training and test errors vs number of trees in ensemble
plot.plot(nTreeList, mseOos)
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Mean Squared Error')
#plot.ylim([0.0, 1.1*max(mseOob)])
plot.show()

# Plot feature importance
featureImportance = wineRFModel.feature_importances_

#scale by max importance
featureImportance = featureImportance / featureImportance.max()
sorted_idx = numpy.argsort(featureImportance)
barPos = numpy.arange(sorted_idx.shape[0]) + .5
plot.barh(barPos, featureImportance[sorted_idx], align='center')
plot.yticks(barPos, wineNames[sorted_idx])
plot.xlabel('Variable Importance')
plot.show()

#printed output
#MSE
#0.314125711509
```

可视化随机森林回归模型的性能

图 7-1 的曲线展示了随机森林算法的减少方差的特性。随着决策树数目的增加，预测误差在下降，曲线的统计波动也在减少。

注意 为了增加对算法的感觉，可以尝试改变代码清单 7-1 中的一些参数，然后观测性能曲线是如何变化的。增加决策树的数目，看看是否可以进一步减少误差，如设置 nTreeList=range(100,1000,100)。改变决策树深度，看看算法对决策树深度是否敏感。红酒数据集大概有 1,600 个实例（行），决策树深度为 10 或 11 个时，每个叶子节点平均含有一个实例左右。当决策树深度为 6 时，就会有 256 个叶子节点，这样每个叶子节点平均有 6 个实例。按照这个范围来调整决策树的深度，观察其对性能的影响。

随机森林可以估计每个属性对预测准确率的贡献（重要性）。代码清单 7-1 提取数据成员 feature_importance（存放了属性对预测结果的贡献），对属性重要性归一化为 0 ～ 1 的数值，按照重要性对属性进行排序，最后形成条状图，如图 7-2 所示。最重要的属性已经归一化为 1.0，并处在条状图的最上面。在随机森林模型中，酒精含量是最重要的属性是不奇怪的。正如在第 5 章看到的，它在惩罚线性回归模型中也是最重要的属性。

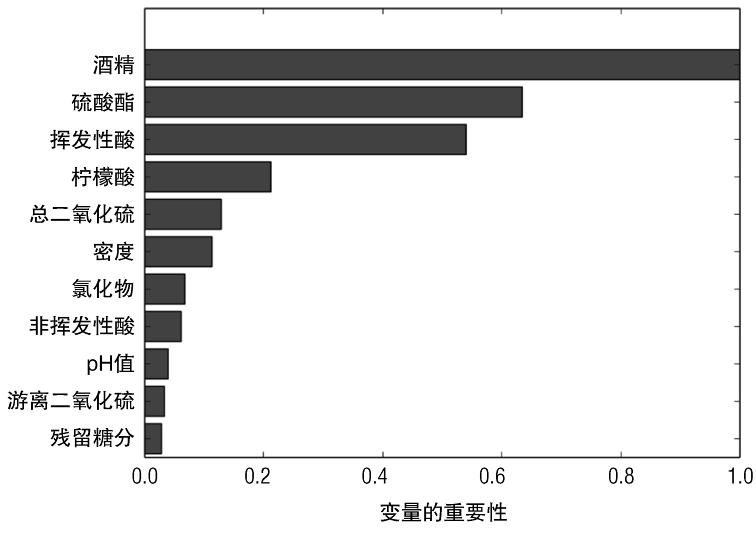


图 7-2 随机森林预测红酒口感，属性的重要性排序

7.1.2 用梯度提升预测红酒品质

如第 6 章所述，梯度提升使用误差最小化的方法来构建决策树，而 Bagging 和随机森

林是使用减少方差的方法。因为梯度提升法使用二元决策树作为基学习器，它也共享了决策树相关的参数。梯度提升法由梯度决定下降的方向，因此还需要步长之类的参数。而且因为梯度提升法采用误差最小化的方法，这也导致对决策树深度的设置具有不同的原理和选择。还可以构建一个随机森林和梯度提升的混合模型，在上层使用梯度提升法，对基学习器使用随机森林的属性随机选择方法。目前只有在 sklearn 集成方法模块中可以看到这种混合。

使用 GradientBoostingRegressor 类的构造函数

下面是 sklearn.ensemble.GradientBoostingRegressor 类的构造函数。

```
class sklearn.ensemble.GradientBoostingRegressor(loss='ls', learning_
rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2, min_
samples_leaf=1, max_depth=3, init=None, random_state=None, max_features=
None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False)
```

下面介绍需要熟悉的参数和方法，以及各种参数的选择及其权衡。

◆ Loss

字符串，可选（缺省值为“ls”）。

梯度提升法用决策树来逼近整体损失函数的梯度。最常使用的整体损失项就是误差平方的和（sum squared error），这个就是通常的最小二乘回归方法的惩罚项。最小误差平方和（Least sum squared error）是一个很方便的选项，因为误差平方（squared error）在数学上处理比较简洁。但是对应实际的问题，其他的损失函数可能更合适。例如，笔者在研究自动交易问题时，注意到误差平方惩罚项会导致算法回避重大的损失，但是会接受较小的损失，但是较小的损失累积起来也是相当可观的。采用误差绝对值的和（Sum of absolute value of error）可以取得更好的性能，对自动交易问题更匹配。最小平均绝对值（Least mean absolute value）通常对异常点不敏感。梯度提升法是少数几个可以自由选择惩罚函数的算法。

可以取的字符串值如下。

- ls ：最小均方误差（Least mean squared error）。
- lad ：最小平均绝对误差（Least mean absolute value of error）。
- huber ：胡贝尔误差是两种误差的混合：当误差数值较小时，取误差的平方，当误差数值较大时，取误差的绝对值。
- quantile ：分位数回归。预测分位数（由 alpha 参数指定）。
- Learning_rate。

浮点数，可选（缺省值为 0.1）。

正如前面提到的，梯度提升法基于梯度下降法。Learning_rate 指明沿梯度方向的步

长。如果步长太大，会看到误差迅速下降，然后迅速上升（是集成方法中决策树数目的函数）。如果步长太小，则误差下降得十分缓慢，需要训练更多的决策树。Learning_rate 的最佳值是依赖于问题的，也依赖于所选择的决策树的深度。缺省值 0.1 是相对比较大的值，但是是一个很好的起点。首先选用这个值，观察是否导致了不稳定或者过拟合，然后再按需调整。

◆ N_estimators

整型，可选（缺省值为 100）。

此参数指定集成方法中的决策树数目。如第 6 章所述，也可以把它看作朝向梯度下降的方向，达到误差最小值所需的步数。也可以看作是增量式逼近所用的步数（即训练模型的数目）。因为每一个后续的逼近(每一个后续的决策树)都与 learning rate(学习速度)相乘，学习速度越大，朝向误差最小值取得同样的进步所需的决策树就越少。然而（正如在学习速度小节所讨论的那样），学习速度太高会导致过拟合。对于一个新问题，往往需要尝试几次才能习得参数的最佳取值范围。缺省值 100 可以作为一个很好的起点（特别是与学习速度的缺省值一起联合使用时）。

◆ Subsample

浮点型，可选（缺省值为 1.0）。

如果与随机森林相似，用数据的抽样对决策树进行训练，则梯度提升法变成了随机梯度提升法。Friedman（算法发明人）建议 subsample 取 0.5。这是一个很好的起点。

◆ Max_depth

整型，可选（缺省值为 3）。

就像随机森林，max_depth 是集成方法中单个决策树的深度。就像第 6 章的简单例子，随机森林需要决策树达到一定深度才能产生高精确度的模型，然而梯度提升通过持续关注残差，使用深度为 1 的决策树（叫作树桩 stumps）就可以获得高精确度。梯度提升法对决策树深度的需求是由属性之间相关程度决定的。如果属性之间相互独立，则深度为 1 的决策树可以获得与深度为 2 的决策树相同的性能。通常，可先将决策树的深度设为 1，然后调整好其他参数。再将决策树的深度调整为 2，看看是否会带来性能上的提升。笔者还从来没遇到过需要决策树深度为 10 的问题。

◆ Max_features

整型、浮点型、字符串，或者 None, 可选（缺省值为 None）。

当查找最佳分割点时，需要考虑的属性的数目是由 max_features 值和问题数据中属性的总数共同决定的。定义属性的总数为 nFeatures，那么：

- ◆ 如果 max_features 是整数，则在每次分割时考虑 max_features 个属性。
- ◆ 如果 max_features 是浮点数，则 max_features 是需要考虑的属性占全体属性

的百分比：int(max_features*nFeatures)。

◆ 可能的字符串值包括：

```
auto max_features=nFeatures
sqrt max_features=sqrt(nFeatures)
log2 max_features=log2(nFeatures)
```

◆ 如果 max_features 是 None，那么 max_features 等于 nFeatures。

在梯度提升法 Python 实现中，max_features 起的作用与随机森林中的作用相同。它决定了在决策树的每个节点进行分割时需要考虑多少个属性。这使梯度提升法的 Python 实现具有一个独特的能力：它可以用随机森林作为基学习器来代替原来需要考虑全部属性空间的决策树。

◆ Warm_start

布尔型，可选（缺省值为 False）。

如果 warm_start 设为 True，fit() 函数将从上次训练停止的地方开始。

下面是用到的属性的描述。

◆ Feature_importances

一个数组，其长度等于数据集中属性的数目。数组中的值是正的浮点数，表明了相应属性对预测结果的重要性。数值越大，表明此属性越重要。

◆ Train_score

一个数组，其长度等于集成方法中决策树的数目。此数组存放在训练阶段对决策树依次训练时的误差。

下面是用到的方法的描述。

◆ Fit(XTrain, yTrain, monitor=None)

XTrain 和 yTrain 的形式与随机森林中的一样。XTrain 是一个（nInstances*nAttributes）numpy 数组，这里 nInstances 是训练数据集的行数，nAttributes 是属性的数目。yTrain 是一个存放训练数据集目标的（nInstances *1）numpy 数组。对象 monitor 是回调函数，用来提早停止训练。

◆ Predict(X)

Predict(x) 由一组属性 X 产生预测，X 的列数（属性数）与训练数据集属性数一致，X 可以有任意行的数据。

◆ Staged_predict(x)

此函数的行为与 predict() 函数的行为类似，除了它是可迭代的，根据梯度提升法生成一系列模型，然后根据模型产生一系列的预测值。每次调用都会利用梯度提升法在已产生的一系列模型中增加一个决策树，然后产生一个预测值。

梯度提升法的参数设置对初学者来说可能有些令人困惑。对梯度提升法参数设置和调整的建议如下。

（1）除了设置 subsample 为 0.5 的情况，其他情况都以缺省值开始训练。模型训练完成后，观察模型在测试数据（out-of-sample，oos）下的预测性能与决策树数目的关系曲线及其变化。

（2）如果测试数据的性能在图的右侧迅速提高，则增加 n_estimators 或者 learning_rate。

（3）如果测试数据的性能在图的右侧迅速恶化，则减少 learning_rate。

（4）一旦测试数据的性能曲线在整体都有些改善（或者只有稍许下降）并且在图的右侧基本持平，则尝试改变 max_depth 和 max_features。

用 GradientBoostingRegressor 实现回归模型

代码清单 7-2 展示了如何针对红酒数据集建立梯度提升模型。

代码清单 7-2 用梯度提升构建回归模型 -wineGBM.py

```
import urllib2
import numpy
from sklearn.cross_validation import train_test_split
from sklearn import ensemble
from sklearn.metrics import mean_squared_error
import pylab as plot

# Read wine quality data from UCI website
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-databases"
"/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
```

```
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

nrows = len(xList)
ncols = len(xList[0])

X = numpy.array(xList)
y = numpy.array(labels)
wineNames = numpy.array(names)

#take fixed holdout set 30% of data rows
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.30,
    random_state=531)

# Train Gradient Boosting model to minimize mean squared error
nEst = 2000
depth = 7
learnRate = 0.01
subSamp = 0.5
wineGBMModel = ensemble.GradientBoostingRegressor(n_estimators=nEst,
                                    max_depth=depth,
                                    learning_rate=learnRate,
                                    subsample = subSamp,
                                    loss='ls')

wineGBMModel.fit(xTrain, yTrain)

# compute mse on test set
msError = []
predictions = wineGBMModel.staged_predict(xTest)
for p in predictions:
    msError.append(mean_squared_error(yTest, p))

print("MSE" )
```

```
print(min(msError))
print(msError.index(min(msError)))

#plot training and test errors vs number of trees in ensemble
plot.figure()
plot.plot(range(1, nEst + 1), wineGBMModel.train_score_,
    label='Training Set MSE')
plot.plot(range(1, nEst + 1), msError, label='Test Set MSE')
plot.legend(loc='upper right')
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Mean Squared Error')
plot.show()

# Plot feature importance
featureImportance = wineGBMModel.feature_importances_

# normalize by max importance
featureImportance = featureImportance / featureImportance.max()
idxSorted = numpy.argsort(featureImportance)
barPos = numpy.arange(idxSorted.shape[0]) + .5
plot.barh(barPos, featureImportance[idxSorted], align='center')
plot.yticks(barPos, wineNames[idxSorted])
plot.xlabel('Variable Importance')
plot.subplots_adjust(left=0.2, right=0.9, top=0.9, bottom=0.1)
plot.show()

# Printed Output:
# for:
#nEst = 2000
#depth = 7
#learnRate = 0.01
#subSamp = 0.5
#
# MSE
# 0.313361215728
# 840
```

代码的第一部分与随机森林的处理过程一样：读取数据集，将属性与目标（标签）分离，

转换为 numpy 数组，然后形成训练和测试数据集。梯度提升的训练序列更简单些。随机森林的代码用一个循环来生成不同 n_estimator 值（即不同决策树数目）的模型，观察测试数据误差与决策树数目的关系。梯度提升法的 Python 实现有一个迭代器（对于回归问题是 staged_predict 函数，对于分类问题是 staged_decision_function 函数）简化了这个过程。使用这些函数，可以训练一个含有 n_estimator 个决策树的模型，并且可以产生所有不同规模（不会超过 n_estimator）下的模型对应的测试数据性能曲线（即利用上述的两个迭代器函数可以生成从 1 到 n_estimator 不同决策树数目下的预测值）。

评估梯度提升模型的性能

图 7-3 和代码清单 7-2 的打印输出结果显示梯度提升法取得了与随机森林同级别的性能，通常是这样。有些问题可能这个方法表现较好，也有可能是另外一个方法表现较好，所以需要两个方法都尝试后才能确定。测试数据误差在图的最右边有轻微的增加，如图 7-3 所示。增加的幅度还需要看具体数值。一般情况下，这种增加还不足以要求减少 learning_rate，然后重新训练。

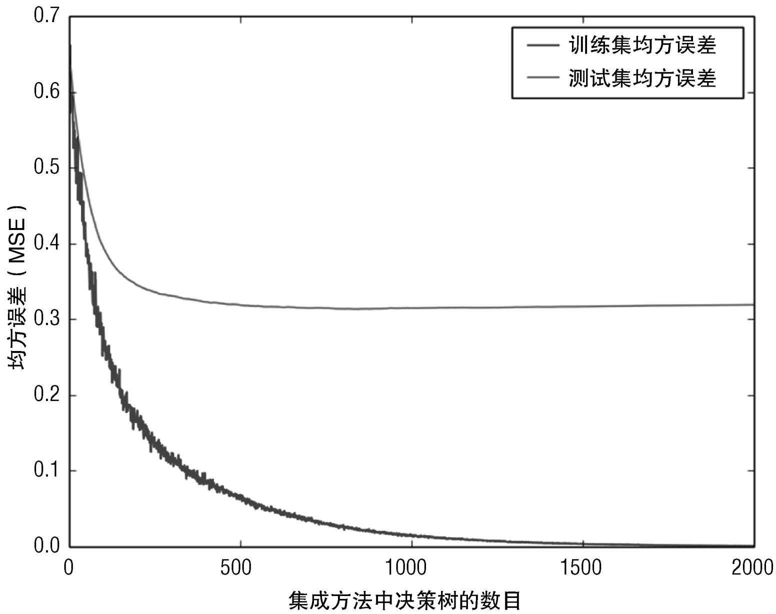


图 7-3　梯度提升法预测红酒口感：均方误差与决策树数目的关系

图 7-4 展示了属性的重要性，梯度提升法实现了此功能。与由随机森林生成的属性重要性比较就可以知道两者很相似，但又不完全相同。两者都认为最重要的属性是酒精含量，在前四或前五最重要的属性排名中，有些属性是一致的。

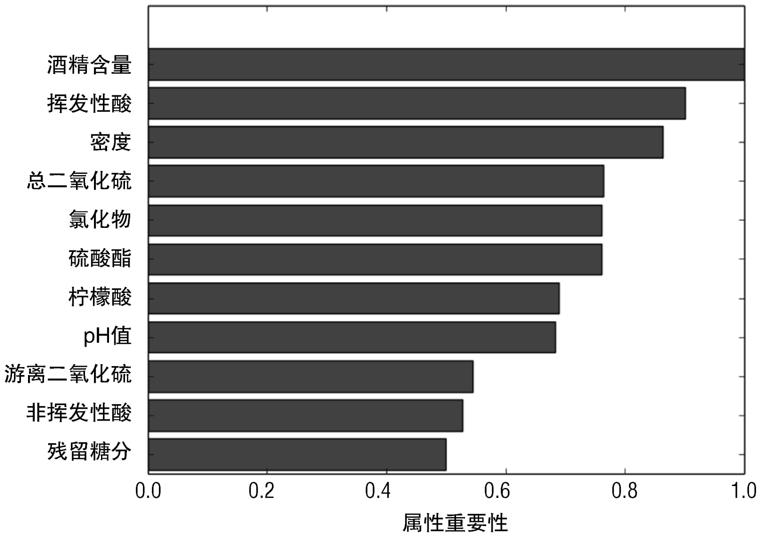


图 7-4 梯度提升法预测红酒口感：各属性相对重要性排序

7.2 用 Bagging 来预测红酒口感

代码清单 7-3 展示了对红酒数据进行自举取样（bootstrap sample），基于取样样本训练决策树，将决策树输出（结果）取平均值作为模型最终的输出（结果），这个叫作 Bagging，这是一个纯粹的减少方差的技术。因此比较 Bagging、随机森林和梯度提升三者的性能是十分有意义的。

代码清单 7-3 对红酒口感问题用 Bagging 构建回归模型（自举集成）-wineBagging.py

```
__author__ = 'mike-bowles'

import urllib2
import numpy
import matplotlib.pyplot as plot
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from math import floor
import random

# Read wine quality data from UCI website
```

```
target_url = ("http://archive.ics.uci.edu/ml/machine-learning-databases"
"/wine-quality/winequality-red.csv")
data = urllib2.urlopen(target_url)

xList = []
labels = []
names = []
firstLine = True
for line in data:
    if firstLine:
        names = line.strip().split(";")
        firstLine = False
    else:
        #split on semi-colon
        row = line.strip().split(";")
        #put labels in separate array
        labels.append(float(row[-1]))
        #remove label from row
        row.pop()
        #convert row to floats
        floatRow = [float(num) for num in row]
        xList.append(floatRow)

nrows = len(xList)
ncols = len(xList[0])


#take fixed test set 30% of sample
nSample = int(nrows * 0.30)
idxTest = random.sample(range(nrows), nSample)
idxTest.sort()
idxTrain = [idx for idx in range(nrows) if not(idx in idxTest)]

#Define test and training attribute and label sets
xTrain = [xList[r] for r in idxTrain]
xTest = [xList[r] for r in idxTest]
yTrain = [labels[r] for r in idxTrain]
```

```
yTest = [labels[r] for r in idxTest]

#train a series of models on random subsets of the training data
#collect the models in a list and check error of composite as list grows

#maximum number of models to generate
numTreesMax = 100

#tree depth - typically at the high end
treeDepth = 5

#initialize a list to hold models
modelList = []
predList = []

#number of samples to draw for stochastic bagging
bagFract = 0.5
nBagSamples = int(len(xTrain) * bagFract)

for iTrees in range(numTreesMax):
    idxBag = []
    for i in range(nBagSamples):
        idxBag.append(random.choice(range(len(xTrain))))
    xTrainBag = [xTrain[i] for i in idxBag]
    yTrainBag = [yTrain[i] for i in idxBag]

    modelList.append(DecisionTreeRegressor(max_depth=treeDepth))
    modelList[-1].fit(xTrainBag, yTrainBag)

    #make prediction with latest model and add to list of predictions
    latestPrediction = modelList[-1].predict(xTest)
    predList.append(list(latestPrediction))


#build cumulative prediction from first "n" models
mse = []
allPredictions = []
```

```
for iModels in range(len(modelList)):

    #average first "iModels" of the predictions
    prediction = []
    for iPred in range(len(xTest)):
        prediction.append(sum([predList[i][iPred] for i in
            range(iModels + 1)])/(iModels + 1))

    allPredictions.append(prediction)
    errors = [(yTest[i] - prediction[i]) for i in range(len(yTest))]
    mse.append(sum([e * e for e in errors]) / len(yTest))

nModels = [i + 1 for i in range(len(modelList))]

plot.plot(nModels,mse)
plot.axis('tight')
plot.xlabel('Number of Models in Ensemble')
plot.ylabel('Mean Squared Error')
plot.ylim((0.0, max(mse)))
plot.show()

print('Minimum MSE')
print(min(mse))


#With treeDepth = 5
#     bagFract = 0.5
#Minimum MSE
#0.429310223079

#With treeDepth = 8
#     bagFract = 0.5
#Minimum MSE
#0.395838627928

#With treeDepth = 10
#     bagFract = 1.0
```

```
#Minimum MSE
#0.313120547589
```

代码清单 7-3 中包含 3 个可调参数。第一个是 numTreesMax，此参数决定构建的决策树数目；第二个是 treeDepth ；第三个是 bagFract。正如第 6 章所讨论的，Bagging 从输入数据中自举取样。这种取样是放回取样，因此有些数据是重复的。参数 bagFract 决定取多少样本。提出此算法的论文建议自举样本数与初始数据集的规模一致，这意味着 bagFract=1.0。代码清单 7-3 产生 numTreesMax 个模型，第一个模型就是第一个决策树，第二个模型就是前二个决策树的平均。第三个模型是前三个决策树的平均，以此类推。代码绘制了均方误差与模型中决策树数目的关系曲线。

图 7-5 和图 7-6 为两种不同参数设置下的结果。Bagging 集成方法应用于红酒口感数据集上，打印结果显示在代码清单的最后。图 7-5 为性能与决策树数目的关系曲线，此时决策树深度为 10，训练用的自举样本规模与初始数据集规模一样（bag fraction = 1.0）。在上述参数设置条件下，Bagging 取得了与随机森林、梯度提升同级别的性能。

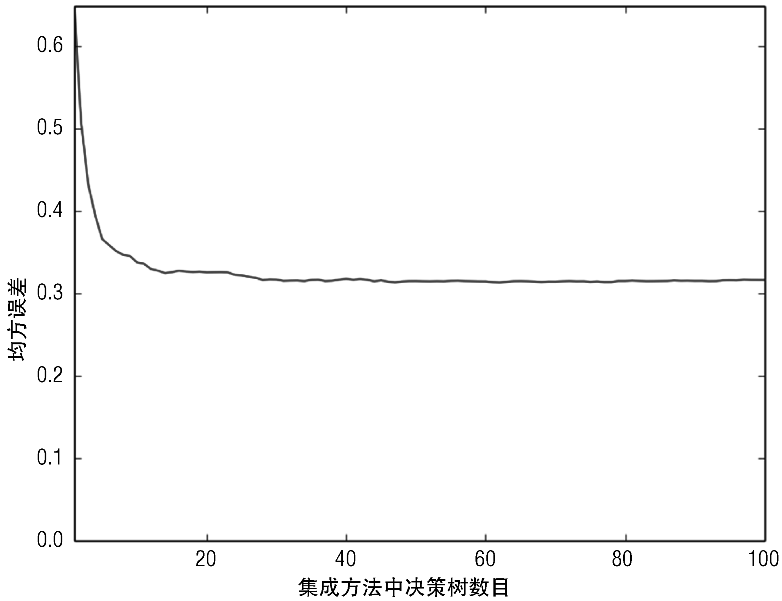


图 7-5 Bagging 方法预测红酒口感：均方误差与决策树数目的关系（决策树深度为 10，bag fraction=1.0）

决策树深度为 8，自举数据规模是初始数据集规模的一半时（bag fraction =0.5），Bagging 方法的性能如图 7-6 所示。如图 7-6 和打印输出的结果所示，此参数选择导致性

能很差。

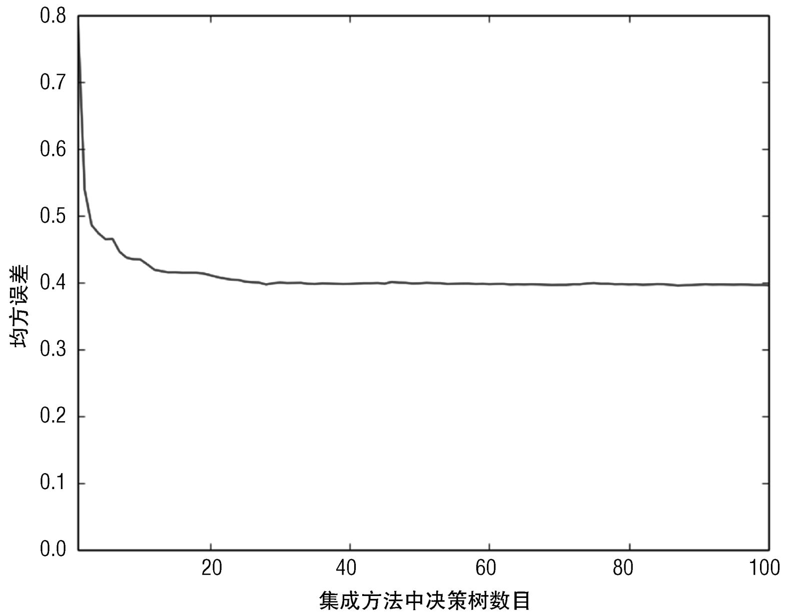


图 7-6　Bagging 方法预测红酒口感：误差与决策树数目的关系（决策树深度为 8，bag fraction=0.5）

7.3　Python 集成方法引入非数值属性

非数值属性是指那些取某几个离散非数值型的属性。人口普查记录就含有大量的非数值属性，“已婚、单身或离异”就是一个例子，家庭住址所在州则是另外一个例子。非数值属性可以提高预测的准确性，但是 Python 集成方法实现需要数值型输入数据。第 4 章和第 5 章介绍了如何对因素变量编码，使其能够引入惩罚线性回归模型。在此也可以采用同样的技术。本节以预测鲍鱼年龄问题为例说明如何应用此项技术。

7.3.1　对鲍鱼性别属性编码引入 Python 随机森林回归方法

假设有一个可以取 n 个值的属性。例如，属性“美国的州”就有 50 个取值，“婚姻状况”有 3 个取值。对于一个有 n 个取值的因素变量，可以创建 $n-1$ 个新的虚拟属性。当变量取第 i 个值时，第 i 个虚拟属性置为 1，其他的虚拟属性置为 0。如果变量取第 n 个值，则所有虚拟变量都置为 0。鲍鱼数据将对此进行详细说明。

代码清单 7-4 展示了利用鲍鱼的重量、壳的尺寸等属性来训练随机森林模型，然后利用模型预测鲍鱼的年龄。此问题的目标是基于各种物理测量指标（鲍鱼各个部分的重量、鲍鱼的尺寸等）来预测鲍鱼的年龄。预测红酒口感的算法也可以用于这

个回归问题。

代码清单 7-4 随机森林方法预测鲍鱼年龄 -abaloneRF.py

```
__author__ = 'mike_bowles'

import urllib2
from pylab import *
import matplotlib.pyplot as plot
import numpy
from sklearn.cross_validation import train_test_split
from sklearn import ensemble
from sklearn.metrics import mean_squared_error


target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/abalone/abalone.data")
#read abalone data
data = urllib2.urlopen(target_url)

xList = []
labels = []
for line in data:
    #split on semi-colon
    row = line.strip().split(",")

    #put labels in separate array and remove label from row
    labels.append(float(row.pop()))

    #form list of list of attributes (all strings)
    xList.append(row)

#code three-valued sex attribute as numeric
xCoded = []
for row in xList:
    #first code the three-valued sex variable
    codedSex = [0.0, 0.0]
    if row[0] == 'M': codedSex[0] = 1.0
    if row[0] == 'F': codedSex[1] = 1.0
```

```
    numRow = [float(row[i]) for i in range(1,len(row))]
    rowCoded = list(codedSex) + numRow
    xCoded.append(rowCoded)
#list of names for
abaloneNames = numpy.array(['Sex1', 'Sex2', 'Length', 'Diameter',
    'Height', 'Whole weight', 'Shucked weight', 'Viscera weight',
    'Shell weight', 'Rings'])

#number of rows and columns in x matrix
nrows = len(xCoded)
ncols = len(xCoded[1])

#form x and y into numpy arrays and make up column names
X = numpy.array(xCoded)
y = numpy.array(labels)

#break into training and test sets.
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.30,
    random_state=531)

#train Random Forest at a range of ensemble sizes in
#order to see how the mse changes
mseOos = []
nTreeList = range(50, 500, 10)
for iTrees in nTreeList:
    depth = None
    maxFeat = 4 #try tweaking
    abaloneRFModel = ensemble.RandomForestRegressor(n_estimators=iTrees,
        max_depth=depth, max_features=maxFeat,
         oob_score=False, random_state=531)

    abaloneRFModel.fit(xTrain,yTrain)

    #Accumulate mse on test set
    prediction = abaloneRFModel.predict(xTest)
    mseOos.append(mean_squared_error(yTest, prediction))
```

```
print("MSE" )
print(mseOos[-1])

#plot training and test errors vs number of trees in ensemble
plot.plot(nTreeList, mseOos)
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Mean Squared Error')
#plot.ylim([0.0, 1.1*max(mseOob)])
plot.show()

# Plot feature importance
featureImportance = abaloneRFModel.feature_importances_

# normalize by max importance
featureImportance = featureImportance / featureImportance.max()
sortedIdx = numpy.argsort(featureImportance)
barPos = numpy.arange(sortedIdx.shape[0]) + .5
plot.barh(barPos, featureImportance[sortedIdx], align='center')
plot.yticks(barPos, abaloneNames[sortedIdx])
plot.xlabel('Variable Importance')
plot.subplots_adjust(left=0.2, right=0.9, top=0.9, bottom=0.1)
plot.show()

# Printed Output:
# MSE
# 4.30971555911
```

此数据集的一个属性就是鲍鱼的性别。鲍鱼的性别有三个可能的取值：雄性、雌性和未成年（一个鲍鱼的性别在幼年期间是不确定的）。因此性别属性是一个三值因素变量。在数据集中，性别属性是 3 个字符变量：*M*、*F* 和 *I*。代码首先定义了一个列表，缺省设置为 2 个浮点型的 0。如果属性值是 *M*，则列表第一个元素的值变为 1.0。如果属性值是 *F*，则列表第二个元素的值变为 1.0。否则，此列表就是 2 个零值（即当属性值是 *I* 时）。然后用这个新的两个元素的列表来代替原来的字符变量，结果用于构建随机森林模型。

7.3.2 评估性能以及变量编码的重要性

图 7-7 展示了随着随机森林集成方法中决策树数目的变化，均方误差是如何减少的。均方误差最小值为 4.31。正如第 2 章关于鲍鱼数据的统计信息所述，鲍鱼年龄（鲍鱼壳上的年轮）的标准差（standard deviation）为 3.22，则其方差为 10.37。因此，针对此数据

集，随机森林大约可以预测年龄方差变化的 58%。

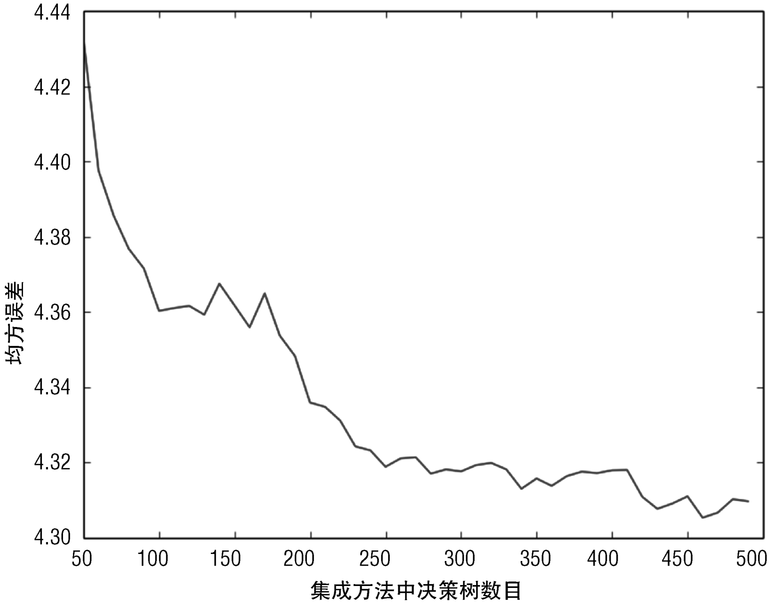


图 7-7 随机森林方法预测鲍鱼年龄：均方误差与决策树数目的关系

对于随机森林模型属性的相对重要性如图 7-8 所示。由性别创建的两个属性：性别属性 1 和性别属性 2 在这个模型中并不是非常重要的。

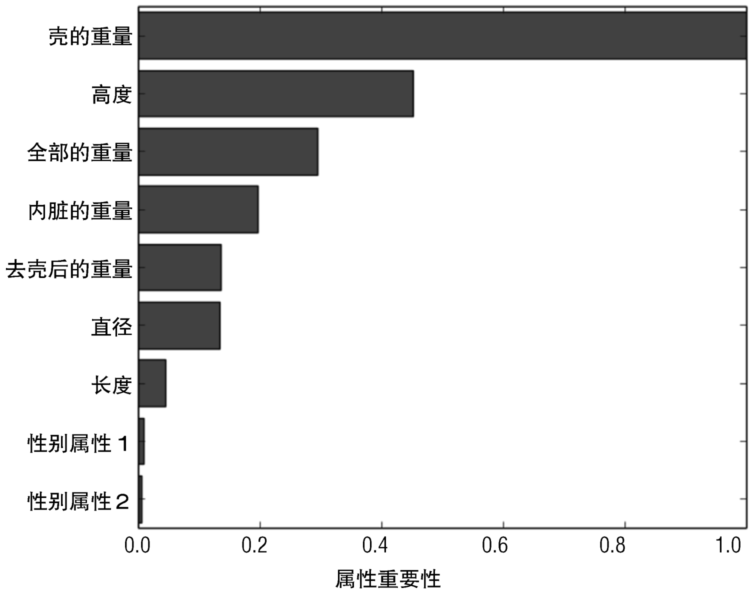


图 7-8 随机森林预测鲍鱼年龄：属性相对重要性排序

7.3.3 在梯度提升回归方法中引入鲍鱼性别属性

梯度提升法对性别变量的处理与随机森林一样。代码清单 7-5 包含了训练梯度提升模型的代码。

代码清单 7-5 用梯度提升法预测鲍鱼年龄 -abaloneGBM.py

```
__author__ = 'mike_bowles'

import urllib2
from pylab import *
import matplotlib.pyplot as plot
import numpy
from sklearn.cross_validation import train_test_split
from sklearn import ensemble
from sklearn.metrics import mean_squared_error

target_url = ("http://archive.ics.uci.edu/ml/machine-learning-"
"databases/abalone/abalone.data")
#read abalone data
data = urllib2.urlopen(target_url)

xList = []
labels = []
for line in data:
    #split on semi-colon
    row = line.strip().split(",")

    #put labels in separate array and remove label from row
    labels.append(float(row.pop()))

    #form list of list of attributes (all strings)
    xList.append(row)

#code three-valued sex attribute as numeric
xCoded = []
for row in xList:
```

```
    #first code the three-valued sex variable
    codedSex = [0.0, 0.0]
    if row[0] == 'M': codedSex[0] = 1.0
    if row[0] == 'F': codedSex[1] = 1.0

    numRow = [float(row[i]) for i in range(1,len(row))]
    rowCoded = list(codedSex) + numRow
    xCoded.append(rowCoded)

#list of names for
abaloneNames = numpy.array(['Sex1', 'Sex2', 'Length', 'Diameter',
    'Height', 'Whole weight', 'Shucked weight',
    'Viscera weight', 'Shell weight', 'Rings'])

#number of rows and columns in x matrix
nrows = len(xCoded)
ncols = len(xCoded[1])

#form x and y into numpy arrays and make up column names
X = numpy.array(xCoded)
y = numpy.array(labels)

#break into training and test sets.
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.30,
    random_state=531)

#instantiate model
nEst = 2000
depth = 5
learnRate = 0.005
maxFeatures = 3
subsamp = 0.5
abaloneGBMModel = ensemble.GradientBoostingRegressor(n_estimators=nEst,
                       max_depth=depth, learning_rate=learnRate,
                        max_features=maxFeatures,subsample=subsamp,
                        loss='ls')
```

```
#train
abaloneGBMModel.fit(xTrain, yTrain)

# compute mse on test set
msError = []
predictions = abaloneGBMModel.staged_decision_function(xTest)
for p in predictions:
    msError.append(mean_squared_error(yTest, p))

print("MSE" )
print(min(msError))
print(msError.index(min(msError)))

#plot training and test errors vs number of trees in ensemble
plot.figure()
plot.plot(range(1, nEst + 1), abaloneGBMModel.train_score_,
    label='Training Set MSE', linestyle=":")
plot.plot(range(1, nEst + 1), msError, label='Test Set MSE')
plot.legend(loc='upper right')
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Mean Squared Error')
plot.show()

# Plot feature importance
featureImportance = abaloneGBMModel.feature_importances_

# normalize by max importance
featureImportance = featureImportance / featureImportance.max()
idxSorted = numpy.argsort(featureImportance)
barPos = numpy.arange(idxSorted.shape[0]) + .5
plot.barh(barPos, featureImportance[idxSorted], align='center')
plot.yticks(barPos, abaloneNames[idxSorted])
plot.xlabel('Variable Importance')
plot.subplots_adjust(left=0.2, right=0.9, top=0.9, bottom=0.1)
plot.show()
# Printed Output:

# for Gradient Boosting
```

```
# nEst = 2000
# depth = 5
# learnRate = 0.003
# maxFeatures = None
# subsamp = 0.5
#
# MSE
# 4.22969363284
# 1736

#for Gradient Boosting with RF base learners
# nEst = 2000
# depth = 5
# learnRate = 0.005
# maxFeatures = 3
# subsamp = 0.5
#
# MSE
# 4.27564515749
# 1687
```

7.3.4 梯度提升法的性能评价以及变量编码的重要性

训练和预测中的注意事项如下。

（1）查看梯度提升法中各属性的重要性排名，看看编码后的性别属性是否重要。

（2）查看将随机森林基学习器引入梯度提升的 Python 实现。这会带来性能提升还是下降？让梯度提升法使用随机森林基学习器只需要改变 max_features 参数，将其从 None 变成一个小于属性总数目的一个整数，或者是小于 1.0 的一个浮点数。当 max_featues 设为 None 时，在决策树生长的过程中，在每个节点寻找最佳的属性以分割数据时，全部 9 个属性是都要考虑的。当 max_features 设为小于 9 的整数，在每个节点进行数据分割时，是随机选择 max_features 个属性来考虑的。

代码清单 7-5 的输出结果显示在代码清单的最后。通过均方误差可以看出随机森林和梯度提升对鲍鱼年龄预测问题来说性能上没有显著的差异。梯度提升法使用简单的决策树还是使用随机森林作为基学习器对性能来说也没有显著的差异。

采用简单的决策树还是随机森林作为基学习器，从预测误差和集成方法规模的关系曲线上来看也几乎没有差别，这个从图 7-9 ～图 7-11 之间的对比就可以看出。

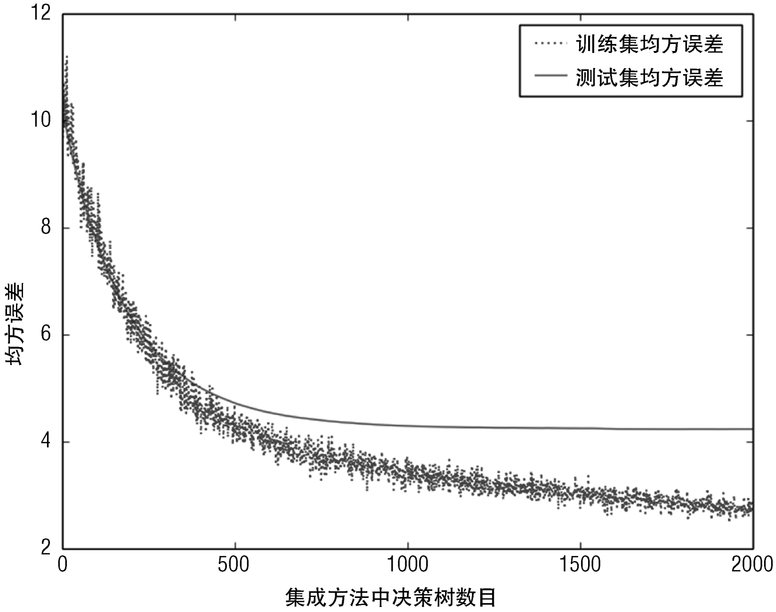


图 7-9 梯度提升法预测鲍鱼年龄：均方误差与决策树数目的关系

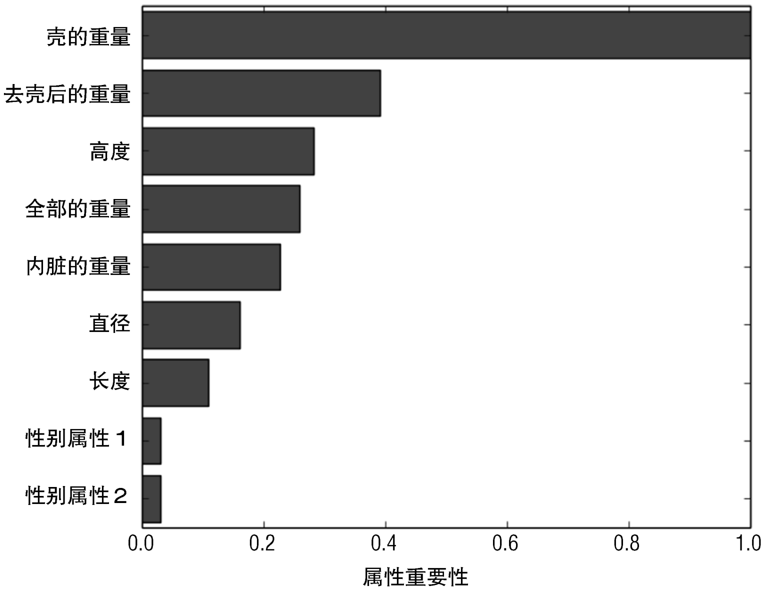


图 7-10 梯度提升法预测鲍鱼年龄：各属性相对重要性排序

图 7-10 和图 7-12 分别展示了基于简单决策树和随机森林的梯度提升法属性的相对重要性的排名。两者之间唯一的差别就是属性“内脏的重量”和“高度”（第 4 和第 5 个最

重要属性）互换了位置。相似地，随机森林中属性的重要性排名和这两个梯度提升法的也几乎没有差别。

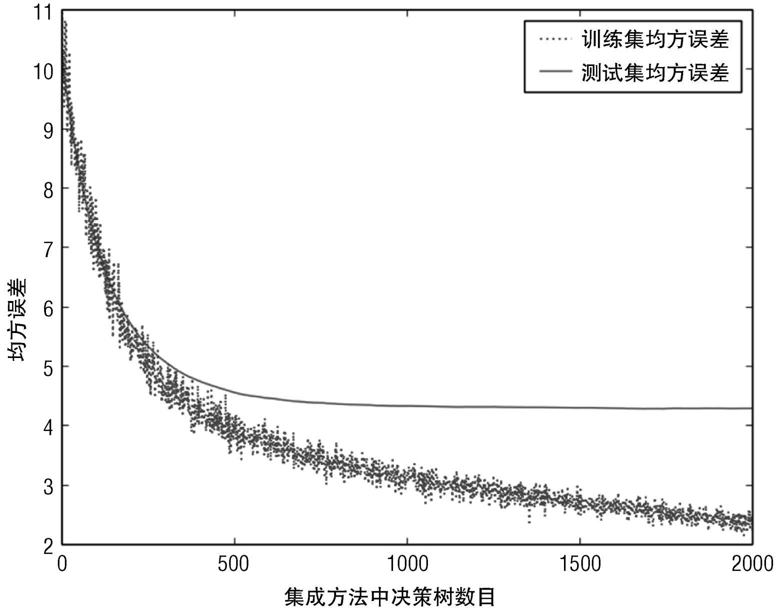


图 7-11　基于随机森林基学习器的梯度提升法预测鲍鱼年龄：均方误差与决策树数目的关系

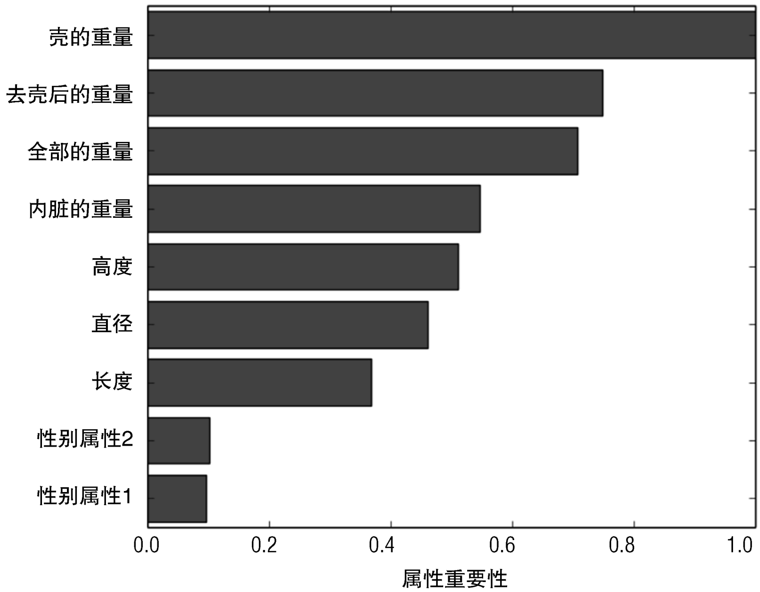


图 7-12　使用随机森林基学习器的梯度提升法预测鲍鱼年龄：各属性相对重要性排名

有看法认为，随机森林适合于规模很大且很稀疏的属性空间，如文本挖掘问题。下节主要比较这两种算法在分类问题上的表现：利用声纳输出将岩石和水雷区分开来。这个问题有 60 个属性，虽然不如文本挖掘问题的属性种类多，但是可能会展示出这两种梯度提升法（基于二元决策树的梯度提升法和基于随机森林基学习器的梯度提升法）在性能上的差异。

7.4 用 Python 集成方法解决二分类问题

此节包含两个基本分类问题：二分类问题和多类别分类问题。二分类问题只有两种输出可能。输出可以是“点击广告”或者“不点击广告”。这里以岩石 vs. 水雷问题为例，利用声纳返回信号判断声纳扫描的是岩石还是水雷。

多类别分类问题是指有超过 2 个可能的输出。用根据玻璃化学成分对玻璃样本进行分类的问题来说明 Python 集成方法是如何解决此类问题的。

7.4.1 用 Python 随机森林方法探测未爆炸的水雷

下面列出了 RandomForestClassifier 类的构造函数及其参数。RandomForestClassifier 的参数绝大多数与 RandomForestRegressor 的一样。RandomForestRegressor 的参数在用 RandomForestRegressor 对红酒口感进行预测的章节中已进行了讨论。这里只强调 RandomForestClassifier 中的不同元素。

第一个不同是用于判断分割点质量的标准。回顾第 6 章中决策树的训练过程，需要尝试所有可能的属性，针对每个属性尝试所有可能的分割点，然后从中找出最佳的属性及其分割点。对于回归决策树，分割点的质量是由平方误差和（sum squared error）决定的。但是平方误差和对于分类问题就不起作用了，需要类似误分类率的指标来描述。

下面是 sklearn.ensemble.RandomForestClassifier 类的构造函数。

```
sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion= 'gini',
max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features='auto',
max_leaf_nodes=None, bootstrap=True, oob_score= False, n_jobs=1, random_
state=None, verbose=0, min_density=None, compute_importances=None)
```

下面是对参数的描述。

◆ Criterion

字符串，可选（缺省值为“gini”）。

可能的取值如下。

Gini ：利用基尼不纯度（Gini impurity）。

Entropy ：利用基于熵的信息增益。

若想获得这两种评价方法的更多信息，可以访问 wikipedia 关于二元决策树的网页（http://en.wikipedia.org/wiki/Decision_tree_learning)。针对目前这个具体实例，这两者对集成方法性能的评价并没有太大的差异。

当训练数据终结于决策树的叶子节点时，叶子节点含有属于不同类别的数据，则根据叶子节点中不同类别的数据所占的百分比，分类决策树自然就可以得到数据属于某个类别的概率。依赖于具体的应用，可能想直接获得上述的概率，或者想直接将叶子节点中所占数据最多的类别作为预测值返回。如果在获得预测结果的同时想要调整阈值，则需要获得概率值。为了生成曲线下面积（area under the curve，AUC)，可能想获得接收者操作特征曲线（receiver operating curve，ROC) 及其概率以保证精确度。如果想计算误分类率，则需要将概率转换为类别的预测。

下面是对方法的描述。

◆ Fit(X, y, sample_weight=None)

随机森林分类版本惟一的不同在于标签 y 的特征。对于分类问题，标签的取值为 0 到类别数减 1 的整数。对于二分类问题，标签的取值是 0 或 1。对于有 nClass 个不同类别的多类别分类问题，标签是从 0 到 nClass-1 的整数。

◆ Predict(X)

对于属性矩阵（二维的 numpy 数组）X，此函数产生所属类别的预测。它生成一个单列的数组，行数等于 X 的行数。每个元素是预测的所属类别，不管问题是二分类问题还是多类别分类问题，都是一样的。

◆ Predict_proba(X)

这个版本的预测函数产生一个二维数组。行数等于 X 的行数。列数就是预测的类别数（对于二分类问题就是 2 列）。每行的元素就是对应类别的概率。

◆ Predict_log_proba(X)

这个版本的预测函数产生一个与 predict_proba 相似的二维数组。但是显示的不是所属类别的概率，而是概率的 log 值。

7.4.2 构建随机森林模型探测未爆炸水雷

代码清单 7-8 展示了如何使用声纳数据构建随机森林模型来预测未爆炸的水雷。数据的预处理和训练的过程与第 6 章和本章前面的随机森林的例子都很相似。不同之处主要在于这是一个分类问题。首先代码将标签从 M 和 R 转换成了 0 和 1。这是 RandomForestClassifer 对输入数据的要求。其次训练完成后，在测试数据集上评价性能阶段。对于二分类问题，评价标准可以选择 ROC 曲线下面积（AUC），或者误分类率。如果可能的话，笔者通常倾向于使用 AUC，因为它能给出整体的性能评价。

为了计算 AUC，使用 predict_proba() 函数。如果预测输出为所属的类别，则不可能获得一个有用的 ROC 曲线（更准确地说，计算出的 ROC 曲线只有 3 个点，两端各 1 个点，中间 1 个点）。sklearn metric 工具包使得 AUC 的计算十分简单，只需要几行代码。代码将结果累积存入一个列表，然后绘出 AUC 性能与决策树数目的关系曲线图。代码清单 7-6 绘制了 AUC 与决策树数目的关系图、30 个最重要属性的相对重要性排序、最大集成规模方法的 ROC 曲线图。代码的最后部分选取 3 个不同的阈值，打印输出针对每个阈值的混淆矩阵（Confusion Matrix）。阈值为 3 个四分位数，结果显示了当阈值从一个分位数转移到另外一个分位数时，假阳性、假阴性是如何变化的。

代码清单 7-6 随机森林分类岩石与水雷 -rocksVMinesRF.py

```
__author__ = 'mike_bowles'

import urllib2
from math import sqrt, fabs, exp
import matplotlib.pyplot as plot
from sklearn.cross_validation import train_test_split
from sklearn import ensemble
from sklearn.metrics import roc_auc_score, roc_curve
import numpy

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []

for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

#separate labels from attributes, convert from attributes from
#string to numeric and convert "M" to 1 and "R" to 0

xNum = []
labels = []
```

```
for row in xList:
    lastCol = row.pop()
    if lastCol == "M":
        labels.append(1)
    else:
        labels.append(0)
    attrRow = [float(elt) for elt in row]
    xNum.append(attrRow)

#number of rows and columns in x matrix
nrows = len(xNum)
ncols = len(xNum[1])

#form x and y into numpy arrays and make up column names
X = numpy.array(xNum)
y = numpy.array(labels)
rocksVMinesNames = numpy.array(['V' + str(i) for i in range(ncols)])

#break into training and test sets.
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.30,
    random_state=531)

auc = []
nTreeList = range(50, 2000, 50)
for iTrees in nTreeList:
    depth = None
    maxFeat = 8 #try tweaking
    rocksVMinesRFModel = ensemble.RandomForestClassifier(n_estimators=
                             iTrees, max_depth=depth, max_features=
                             maxFeat, oob_score=False, random_state=531)
                             rocksVMinesRFModel.fit(xTrain,yTrain)

    #Accumulate auc on test set
    prediction = rocksVMinesRFModel.predict_proba(xTest)
    aucCalc = roc_auc_score(yTest, prediction[:,1:2])
    auc.append(aucCalc)

print("AUC" )
print(auc[-1])
```

```
#plot training and test errors vs number of trees in ensemble
plot.plot(nTreeList, auc)
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Area Under ROC Curve - AUC')
#plot.ylim([0.0, 1.1*max(mseOob)])
plot.show()

# Plot feature importance
featureImportance = rocksVMinesRFModel.feature_importances_

# normalize by max importance
featureImportance = featureImportance / featureImportance.max()

#plot importance of top 30
idxSorted = numpy.argsort(featureImportance)[30:60]
idxTemp = numpy.argsort(featureImportance)[::-1]
print(idxTemp)
barPos = numpy.arange(idxSorted.shape[0]) + .5
plot.barh(barPos, featureImportance[idxSorted], align='center')
plot.yticks(barPos, rocksVMinesNames[idxSorted])
plot.xlabel('Variable Importance')
plot.show()

#plot best version of ROC curve
fpr, tpr, thresh = roc_curve(yTest, list(prediction[:,1:2]))
ctClass = [i*0.01 for i in range(101)]

plot.plot(fpr, tpr, linewidth=2)
plot.plot(ctClass, ctClass, linestyle=':')
plot.xlabel('False Positive Rate')
plot.ylabel('True Positive Rate')
plot.show()

#pick some threshold values and calc confusion matrix for
#best predictions

#notice that GBM predictions don't fall in range of (0, 1)
#pick threshold values at 25th, 50th and 75th percentiles
```

```
idx25 = int(len(thresh) * 0.25)
idx50 = int(len(thresh) * 0.50)
idx75 = int(len(thresh) * 0.75)

#calculate total points, total positives and total negatives
totalPts = len(yTest)
P = sum(yTest)
N = totalPts - P

print('')
print('Confusion Matrices for Different Threshold Values')

#25th
TP = tpr[idx25] * P; FN = P - TP; FP = fpr[idx25] * N; TN = N - FP
print('')
print('Threshold Value = ', thresh[idx25])
print('TP = ', TP/totalPts, 'FP = ', FP/totalPts)
print('FN = ', FN/totalPts, 'TN = ', TN/totalPts)

#50th
TP = tpr[idx50] * P; FN = P - TP; FP = fpr[idx50] * N; TN = N - FP
print('')
print('Threshold Value = ', thresh[idx50])
print('TP = ', TP/totalPts, 'FP = ', FP/totalPts)
print('FN = ', FN/totalPts, 'TN = ', TN/totalPts)

#75th
TP = tpr[idx75] * P; FN = P - TP; FP = fpr[idx75] * N; TN = N - FP
print('')
print('Threshold Value = ', thresh[idx75])
print('TP = ', TP/totalPts, 'FP = ', FP/totalPts)
print('FN = ', FN/totalPts, 'TN = ', TN/totalPts)

# Printed Output:
#
# AUC
# 0.950304259635
#
# Confusion Matrices for Different Threshold Values
#
```

```
# ('Threshold Value = ', 0.76051282051282054)
# ('TP = ', 0.25396825396825395, 'FP = ', 0.0)
# ('FN = ', 0.2857142857142857, 'TN = ', 0.46031746031746029)
#
# ('Threshold Value = ', 0.62461538461538457)
# ('TP = ', 0.46031746031746029, 'FP = ', 0.047619047619047616)
# ('FN = ', 0.079365079365079361, 'TN = ', 0.41269841269841268)
#
# ('Threshold Value = ', 0.46564102564102566)
# ('TP = ', 0.53968253968253965, 'FP = ', 0.22222222222222221)
# ('FN = ', 0.0, 'TN = ', 0.23809523809523808)
```

7.4.3 随机森林分类器的性能

图 7-13 为 AUC 与决策树数目的关系图。此图是以往看过的均方误差或误分类错误与决策树数目关系图的倒置。对于均方误差和误分类错误,值越小越好。对于 AUC 来说,1.0 是相当完美的，但是 0.5 就很差了。因此，AUC 值越大越好，不是寻找曲线中的波谷，而是找波峰。图 7-13 展示在曲线的左侧出现了一个波峰。然而,因为随机森林只减少方差,不会过拟合，因此波峰也可以归功于随机的波动。正像本章早期的回归问题，最佳模型是包含所有决策树的模型，其性能是图中最右的点。

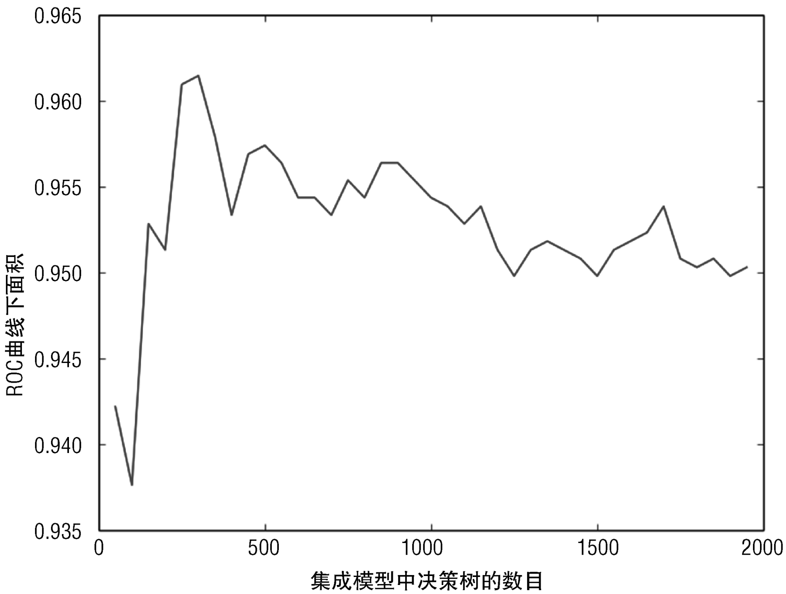


图 7-13 随机森林探测未爆炸水雷：AUC 与决策树数目的关系

图 7-14 是最重要的前 30 个属性的重要性排名。在水雷检测的问题中，不同的属性对应不同频率的声纳信号，即不同波长的信号。如果要求设计一个机器学习系统来解决这个问题，下一步就是确定这些属性对应的波长，确定波长与水雷和岩石在不同维度上的对应关系。这样可以加深对模型的理解。

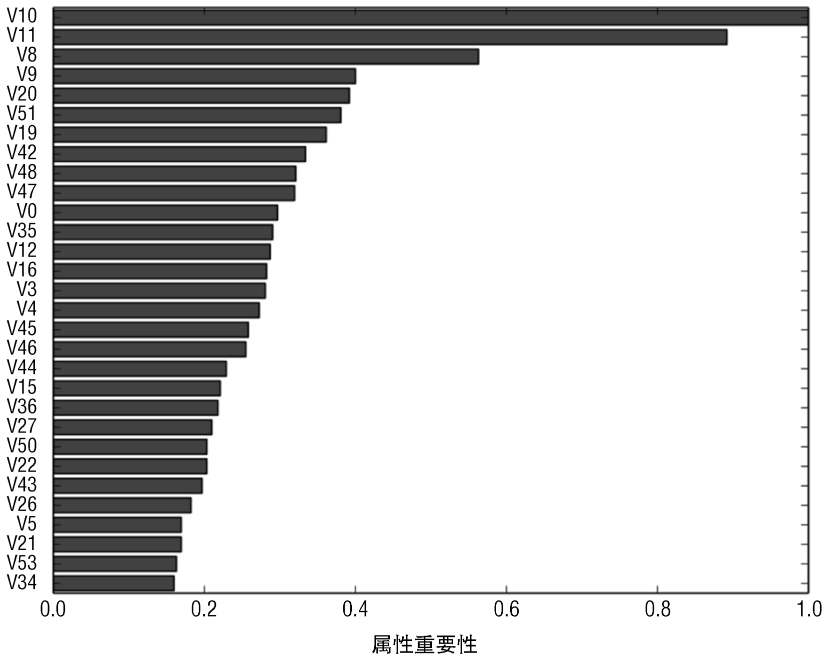


图 7-14　随机森林探测未爆炸水雷：各属性相对重要性排名

此模型的 AUC 值相当高，其 ROC 曲线表现也相当好。虽然还达不到左上角，但也十分接近。

7.4.4　用 Python 梯度提升法探测未爆炸水雷

代码清单 7-7 为 scikitlearn 中梯度提升法的构造函数。绝大多数 GradientBoosting Classifer 的参数和方法与 GradientBoostingRegressor 的相同。下面主要描述其不同的参数和方法。

下面是 sklearn.ensemble.GradientBoostingClassifier 类的构造函数。

```
sklearn.ensemble.GradientBoostingClassifier(loss='deviance', learning_
rate=0.1, n_estimators=100, subsample=1.0, min_samples_split=2,
min_samples_leaf=1, max_depth=3, init=None, random_state=None,
max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False)
```

下面是对参数的描述。

◆ loss

deviance 对于分类问题，deviance 是缺省的，也是唯一的选项。

下面是对方法的描述。

◆ fit（X,y , monitor=None)

对于分类问题，其不同点只在于标签 y 的不同。对应分类问题，标签是 0 到类别总数减 1 的一个整数。对于二分类问题，标签值为 0 或者 1。对于多类别分类问题，如果共有 nClass 个不同的类别，则标签取值为 0 ～ nClass-1。

◆ decision_function(X)

梯度提升分类器实际上是回归决策树的集合，会产生与所属类别的概率相关的实数估计值。这些估计值还需要经过反 logistic 函数将其转换为概率。转换前的实数估计值可通过此函数获得，对这些估计值的使用就像 ROC 曲线计算中使用概率那样简单。

◆ predict(X)

此函数预测所属类别。

◆ predict_proba(X)

此函数预测所属类别的概率。它对于每个类别有一列概率值。对于二分类问题有两列。对于多类别分类问题，共有 nClass 列。

上述函数的阶段性（staged）版本是可迭代的，产生与决策树数目相同的结果（也与训练过程中执行的步数一致）。

◆ staged_decison_function(X)

此为 decision 函数的可迭代版本。

◆ staged_predict(X)

此为 predict 函数的可迭代版本。

◆ staged_predict_proba(X)

此为 predict_proba 函数的可迭代版本。

代码清单 7-7 使用 sklearn 的 GradientBoostingClassifier 完成探测未爆炸水雷的任务。

代码清单 7-7 梯度提升法分类岩石与水雷 -rocksVMinesGBM.py

```
__author__ = 'mike_bowles'

import urllib2
from math import sqrt, fabs, exp
import matplotlib.pyplot as plot
```

```
from sklearn.cross_validation import train_test_split
from sklearn import ensemble
from sklearn.metrics import roc_auc_score, roc_curve
import numpy

#read data from uci data repository
target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/undocumented/connectionist-bench/sonar/sonar.all-data")
data = urllib2.urlopen(target_url)


#arrange data into list for labels and list of lists for attributes
xList = []


for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

#separate labels from attributes, convert from attributes from
#string to numeric and convert "M" to 1 and "R" to 0

xNum = []
labels = []

for row in xList:
    lastCol = row.pop()
    if lastCol == "M":
        labels.append(1)
    else:
        labels.append(0)
    attrRow = [float(elt) for elt in row]
    xNum.append(attrRow)

#number of rows and columns in x matrix
nrows = len(xNum)
ncols = len(xNum[1])

#form x and y into numpy arrays and make up column names
```

```
X = numpy.array(xNum)
y = numpy.array(labels)
rockVMinesNames = numpy.array(['V' + str(i) for i in range(ncols)])

#break into training and test sets.
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=0.30,
    random_state=531)

#instantiate model
nEst = 2000
depth = 3
learnRate = 0.007
maxFeatures = 20
rockVMinesGBMModel = ensemble.GradientBoostingClassifier(
                        n_estimators=nEst, max_depth=depth,
                        learning_rate=learnRate,
                        max_features=maxFeatures)
#train
rockVMinesGBMModel.fit(xTrain, yTrain)

# compute auc on test set as function of ensemble size
auc = []
aucBest = 0.0
predictions = rockVMinesGBMModel.staged_decision_function(xTest)
for p in predictions:
    aucCalc = roc_auc_score(yTest, p)
    auc.append(aucCalc)

    #capture best predictions
    if aucCalc > aucBest:
        aucBest = aucCalc
        pBest = p

    idxBest = auc.index(max(auc))

    #print best values
    print("Best AUC" )
    print(auc[idxBest])
    print("Number of Trees for Best AUC")
    print(idxBest)
```

```
    #plot training deviance and test auc's vs number of trees in ensemble
    plot.figure()
    plot.plot(range(1, nEst + 1), rockVMinesGBMModel.train_score_,
        label='Training Set Deviance', linestyle=":")
    plot.plot(range(1, nEst + 1), auc, label='Test Set AUC')
    plot.legend(loc='upper right')
    plot.xlabel('Number of Trees in Ensemble')
    plot.ylabel('Deviance / AUC')
    plot.show()

# Plot feature importance
featureImportance = rockVMinesGBMModel.feature_importances_

# normalize by max importance
featureImportance = featureImportance / featureImportance.max()

#plot importance of top 30
idxSorted = numpy.argsort(featureImportance)[30:60]

barPos = numpy.arange(idxSorted.shape[0]) + .5
plot.barh(barPos, featureImportance[idxSorted], align='center')
plot.yticks(barPos, rockVMinesNames[idxSorted])
plot.xlabel('Variable Importance')
plot.show()

#pick threshold values and calc confusion matrix for best predictions
#notice that GBM predictions don't fall in range of (0, 1)

#plot best version of ROC curve
fpr, tpr, thresh = roc_curve(yTest, list(pBest))
ctClass = [i*0.01 for i in range(101)]

plot.plot(fpr, tpr, linewidth=2)
plot.plot(ctClass, ctClass, linestyle=':')
plot.xlabel('False Positive Rate')
plot.ylabel('True Positive Rate')
plot.show()

#pick threshold values and calc confusion matrix for best predictions
```

```
#notice that GBM predictions don't fall in range of (0, 1)
#pick threshold values at 25th, 50th and 75th percentiles
idx25 = int(len(thresh) * 0.25)
idx50 = int(len(thresh) * 0.50)
idx75 = int(len(thresh) * 0.75)

#calculate total points, total positives and total negatives
totalPts = len(yTest)
P = sum(yTest)
N = totalPts - P

print('')
print('Confusion Matrices for Different Threshold Values')

#25th
TP = tpr[idx25] * P; FN = P - TP; FP = fpr[idx25] * N; TN = N - FP
print('')
print('Threshold Value = ', thresh[idx25])
print('TP = ', TP/totalPts, 'FP = ', FP/totalPts)
print('FN = ', FN/totalPts, 'TN = ', TN/totalPts)

#50th
TP = tpr[idx50] * P; FN = P - TP; FP = fpr[idx50] * N; TN = N - FP
print('')
print('Threshold Value = ', thresh[idx50])
print('TP = ', TP/totalPts, 'FP = ', FP/totalPts)
print('FN = ', FN/totalPts, 'TN = ', TN/totalPts)

#75th
TP = tpr[idx75] * P; FN = P - TP; FP = fpr[idx75] * N; TN = N - FP
print('')
print('Threshold Value = ', thresh[idx75])
print('TP = ', TP/totalPts, 'FP = ', FP/totalPts)
print('FN = ', FN/totalPts, 'TN = ', TN/totalPts)


# Printed Output:
#
# Best AUC
# 0.936105476673
```

```
# Number of Trees for Best AUC
# 1989
#
# Confusion Matrices for Different Threshold Values
#
# ('Threshold Value = ', 6.2941249291909935)
# ('TP = ', 0.23809523809523808, 'FP = ', 0.015873015873015872)
# ('FN = ', 0.30158730158730157, 'TN = ', 0.44444444444444442)
#
# ('Threshold Value = ', 2.2710265370949441)
# ('TP = ', 0.44444444444444442, 'FP = ', 0.063492063492063489)
# ('FN = ', 0.095238095238095233, 'TN = ', 0.3968253968253968)
#
# ('Threshold Value = ', -3.0947902666953317)
# ('TP = ', 0.53968253968253965, 'FP = ', 0.22222222222222221)
# ('FN = ', 0.0, 'TN = ', 0.23809523809523808)
#
#
# Printed Output with max_features = 20 (Random Forest base learners):
#
# Best AUC
# 0.956389452333
# Number of Trees for Best AUC
# 1426
#
# Confusion Matrices for Different Threshold Values
#
# ('Threshold Value = ', 5.8332200248698536)
# ('TP = ', 0.23809523809523808, 'FP = ', 0.015873015873015872)
# ('FN = ', 0.30158730158730157, 'TN = ', 0.44444444444444442)
#
# ('Threshold Value = ', 2.0281780133610567)
# ('TP = ', 0.47619047619047616, 'FP = ', 0.031746031746031744)
# ('FN = ', 0.063492063492063489, 'TN = ', 0.42857142857142855)
#
# ('Threshold Value = ', -1.2965629080181333)
# ('TP = ', 0.53968253968253965, 'FP = ', 0.22222222222222221)
# ('FN = ', 0.0, 'TN = ', 0.23809523809523808)
```

此代码的流程与随机森林的基本相同。一个不同点是梯度提升法会过拟合，因此当

代码累积计算 AUC 并将其存入一个列表准备绘制时，程序同时持续跟踪 AUC 的最佳值。其最佳值用来生成 ROC 曲线、假阳性、假阴性等指标（如图 7-15 所示）。另一个不同是梯度提升方法运行了 2 次，一次是使用普通的决策树，另外一次是使用随机森林基学习器。两者都获得了很好的分类性能。使用随机森林基学习器的性能更好，这点与鲍鱼预测年龄问题不同。在鲍鱼预测年龄问题上，采用随机森林基学习器并没有获得显著的改善。

7.4.5 梯度提升法分类器的性能

图 7-16 绘制了两个曲线。一个是训练集的偏差（deviance）。偏差与估计值距离正确值有多远相关，但又与误分类误差稍微有些差别。把偏差绘制出来是因为梯度提升法就是把偏差作为优化的目标。在图中可以看到训练过程中偏差的变化。AUC（基于测试数据）也在图中绘出是为了展示随着决策树数目的增加测试数据的性能的变化（决策树数目的增加就等同于在梯度下降中采取了更多的步骤；每一步就意味着又训练了一个决策树）。

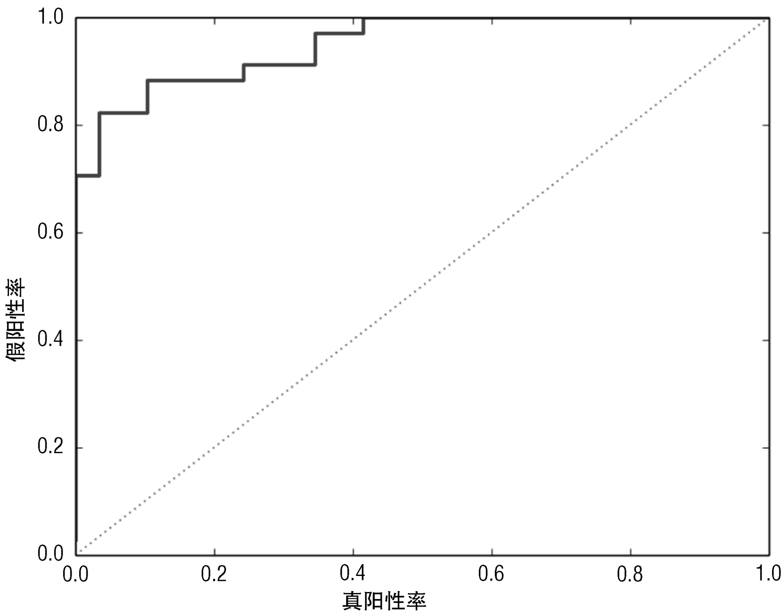


图 7-15 随机森林模型探测未爆炸水雷：ROC 曲线

图 7-17 绘制了最重要的前 30 个属性的重要性排名。图 7-17 中的属性重要性排名与随机森林的（见图 7-14）有些不同，但是也有共同点，如属性 V10、V11、V20 和 V51 排名都比较靠前，尽管具体的顺序可能不同。

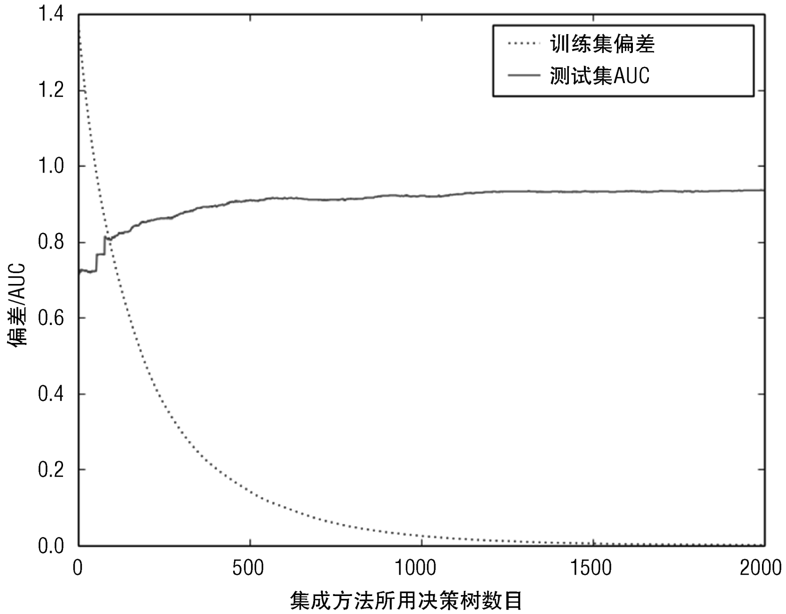


图 7-16　梯度提升模型探测未爆炸水雷：AUC 与集成方法规模的关系

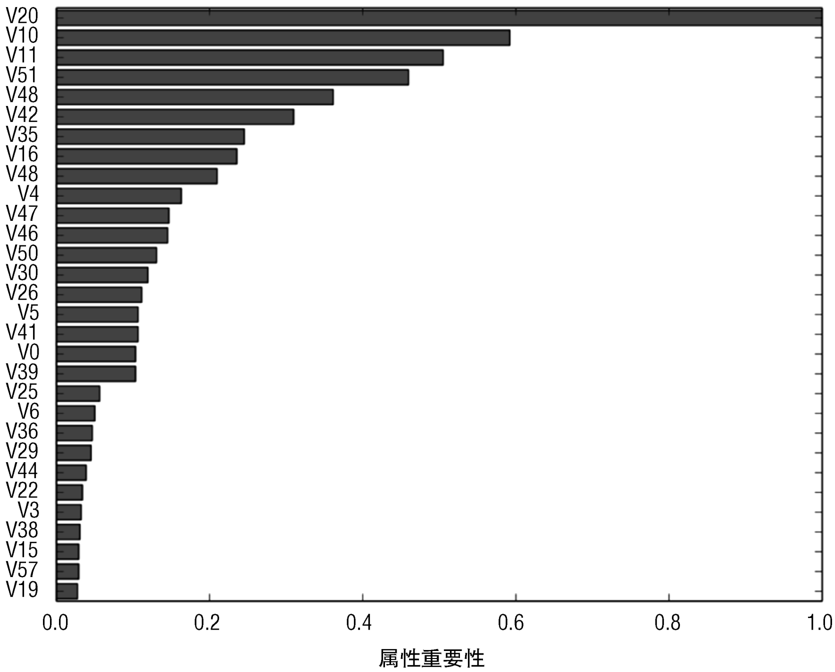


图 7-17　梯度提升模型探测未爆炸水雷：各属性重要性排名

使用随机森林基学习器的梯度提升法的模型训练过程如图 7-18，7-19 所示。使用随机森林基学习器确实获得了更好的结果，但是这种差异还不足以在图中有所体现。

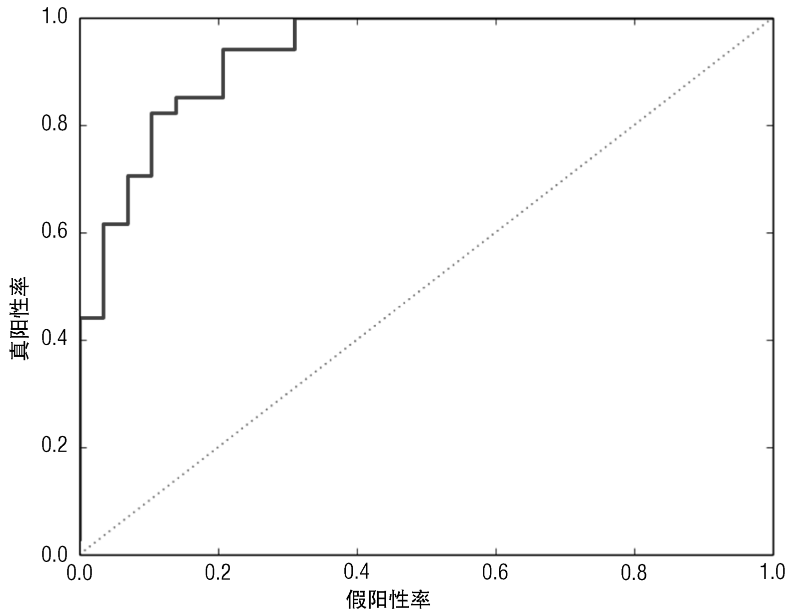


图 7-18 梯度提升法探测未爆炸水雷：ROC 曲线

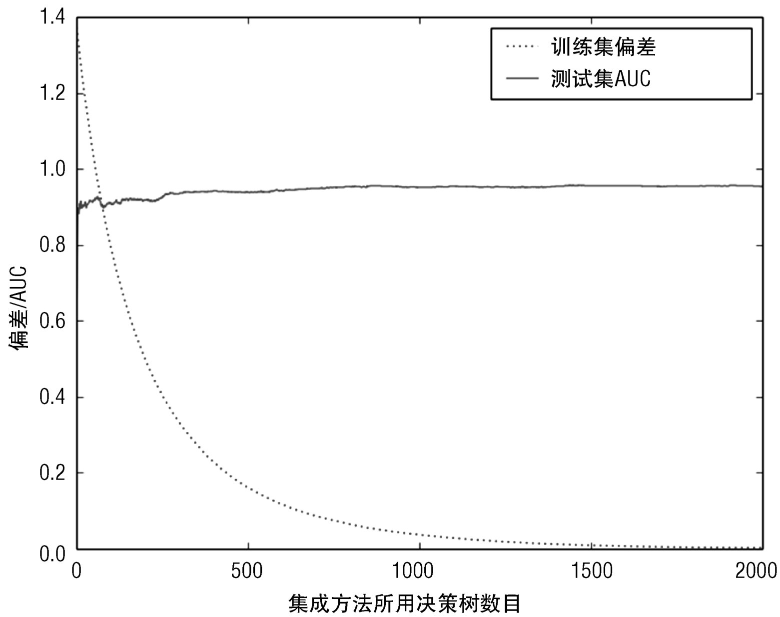


图 7-19 使用随机森林基学习器的梯度提升法检测水雷：AUC 与集成方法规模的关系

通过比较图 7-20 与图 7-17 可知使用随机森林基学习器对属性的重要性影响不大。

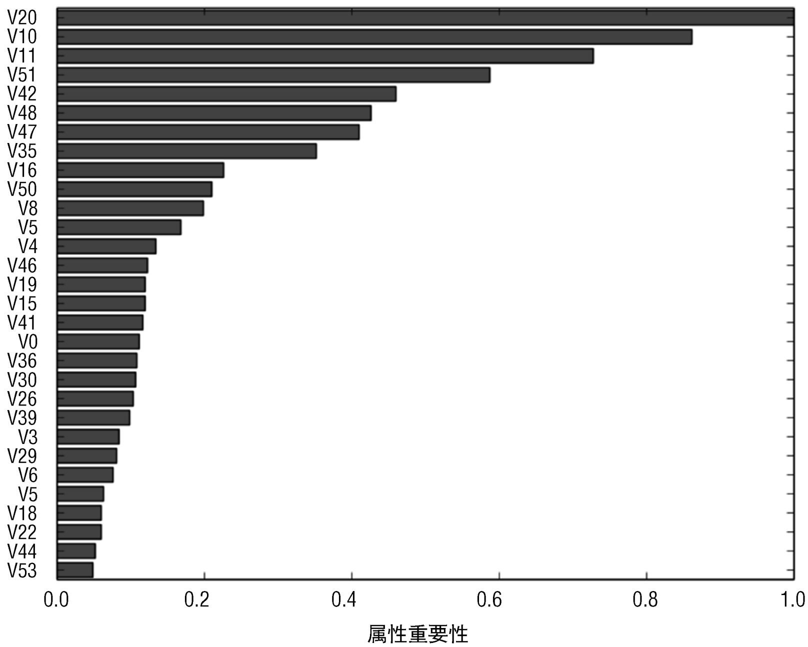


图 7-20 使用随机森林基学习器的梯度提升法探测水雷：各属性重要性排名

图 7-21 为使用随机森林基学习器的梯度提升法检测水雷的 ROC 曲线。

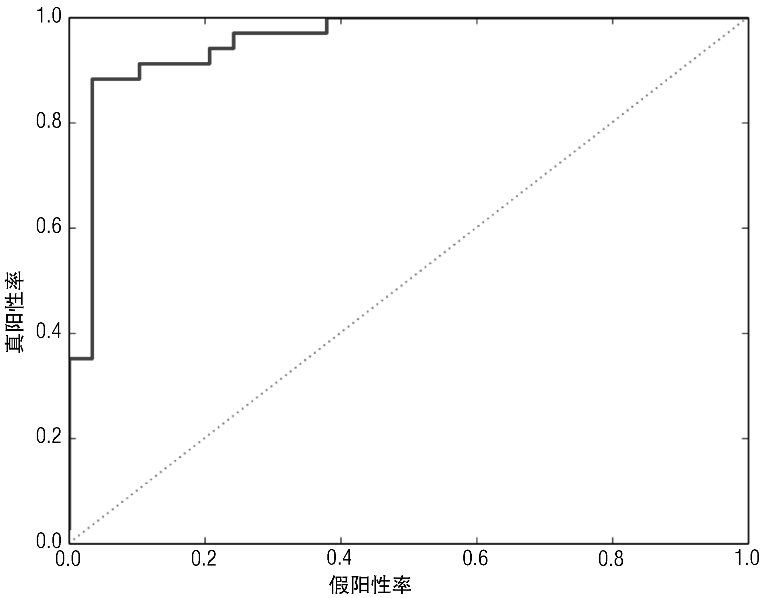


图 7-21 使用随机森林基学习器的梯度提升法探测水雷：ROC 曲线

此节介绍了如何用集成方法解决二分类问题。在很多方面，使用集成方法解决二分类问题与回归问题基本相同，可以注意到初始化 RandomForestRegressor 所需的参数与初始化 RandomForestClassification 的基本一样。基于第 6 章的介绍，就可以理解这种一致背后的原因。

集成方法用于分类问题和回归问题的不同之处主要在于对误差的评价方法和误差特征的刻画不同。

下节主要介绍上述方法如何解决多类别分类问题。

7.5 用 Python 集成方法解决多类别分类问题

Python 实现的随机森林和梯度提升可以构建二分类和多类别分类模型。这两个模型之间有天然的差别。首先是标签（y）。随机森林和梯度提升的讨论就包含对标签的处理。对于一个有 nClass 个不同类别的分类问题，标签取 0 ～ nClass-1 的整数。另外一个体现类别数量的是不同预测方法的输出。预测所属类别的方法输出与标签相同范围的整数值，预测所属类别概率的方法输出为 nClass 个可能类别对应的概率。

另外一个需要关注差异性的领域就是对性能的评价。误分类错误仍然是有意义的，在此节将看到基于误分类错误评价测试数据性能的代码。当类别超过两个时，AUC 的使用将更复杂，不同误差类型之间的权衡也将更有挑战性。

7.5.1 用随机森林对玻璃进行分类

代码清单 7-8 遵循的流程与用于探测水雷的代码基本一致。

代码清单 7-8 用随机森林对玻璃进行分类 -glassRF.py

```
__author__ = 'mike_bowles'

import urllib2
from math import sqrt, fabs, exp
import matplotlib.pyplot as plot
from sklearn.linear_model import enet_path
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve
from sklearn.cross_validation import train_test_split
from sklearn import ensemble
import numpy

target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/glass/glass.data")
```

```
data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []
for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

glassNames = numpy.array(['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca',
    'Ba', 'Fe', 'Type'])

#Separate attributes and labels
xNum = []
labels = []

for row in xList:
    labels.append(row.pop())
    l = len(row)
    #eliminate ID
    attrRow = [float(row[i]) for i in range(1, l)]
    xNum.append(attrRow)

#number of rows and columns in x matrix
nrows = len(xNum)
ncols = len(xNum[1])

#Labels are integers from 1 to 7 with no examples of 4.
#gb requires consecutive integers starting at 0
newLabels = []
labelSet = set(labels)
labelList = list(labelSet)
labelList.sort()
nlabels = len(labelList)
for l in labels:
    index = labelList.index(l)
    newLabels.append(index)

#Class populations:
```

```
#old label     new label         num of examples
#1              0                   70
#2              1                   76
#3              2                   17
#5              3                   13
#6              4                   9
#7              5                   29
#
#Drawing 30% test sample may not preserve population proportions

#stratified sampling by labels.
xTemp = [xNum[i] for i in range(nrows) if newLabels[i] == 0]
yTemp = [newLabels[i] for i in range(nrows) if newLabels[i] == 0]
xTrain, xTest, yTrain, yTest = train_test_split(xTemp, yTemp,
    test_size=0.30, random_state=531)
for iLabel in range(1, len(labelList)):
    #segregate x and y according to labels
    xTemp = [xNum[i] for i in range(nrows) if newLabels[i] == iLabel]
    yTemp = [newLabels[i] for i in range(nrows) if \
        newLabels[i] == iLabel]

    #form train and test sets on segregated subset of examples
    xTrainTemp, xTestTemp, yTrainTemp, yTestTemp = train_test_split(
        xTemp, yTemp, test_size=0.30, random_state=531)

    #accumulate
    xTrain = numpy.append(xTrain, xTrainTemp, axis=0)
    xTest = numpy.append(xTest, xTestTemp, axis=0)
    yTrain = numpy.append(yTrain, yTrainTemp, axis=0)
    yTest = numpy.append(yTest, yTestTemp, axis=0)

missCLassError = []
nTreeList = range(50, 2000, 50)
for iTrees in nTreeList:
    depth = None
    maxFeat = 4 #try tweaking
    glassRFModel = ensemble.RandomForestClassifier(n_estimators=iTrees,
                      max_depth=depth, max_features=maxFeat,
                      oob_score=False, random_state=531)
```

```
    glassRFModel.fit(xTrain,yTrain)

    #Accumulate auc on test set
    prediction = glassRFModel.predict(xTest)
    correct = accuracy_score(yTest, prediction)

    missCLassError.append(1.0 - correct)

print("Missclassification Error" )
print(missCLassError[-1])

#generate confusion matrix
pList = prediction.tolist()
confusionMat = confusion_matrix(yTest, pList)
print('')
print("Confusion Matrix")
print(confusionMat)

#plot training and test errors vs number of trees in ensemble
plot.plot(nTreeList, missCLassError)
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Missclassification Error Rate')
#plot.ylim([0.0, 1.1*max(mseOob)])
plot.show()

# Plot feature importance
featureImportance = glassRFModel.feature_importances_

# normalize by max importance
featureImportance = featureImportance / featureImportance.max()

#plot variable importance
idxSorted = numpy.argsort(featureImportance)
barPos = numpy.arange(idxSorted.shape[0]) + .5
plot.barh(barPos, featureImportance[idxSorted], align='center')
plot.yticks(barPos, glassNames[idxSorted])
plot.xlabel('Variable Importance')
plot.show()
```

```
# Printed Output:
# Missclassification Error
# 0.227272727273
#
# Confusion Matrix
# [[17  1  2  0  0  1]
#  [ 2 18  1  2  0  0]
#  [ 3  0  3  0  0  0]
#  [ 0  0  0  4  0  0]
#  [ 0  1  0  0  2  0]
#  [ 0  2  0  0  0  7]]
```

7.5.2 处理类不均衡问题

此段代码遵循的流程与用于探测水雷的代码基本一致，主要有以下的不同。从代码中可以看到，将原始数据中的一系列不同玻璃类型转换为相应的整数，主要是为了满足随机森林的输入要求。代码还显示了不同类别玻璃的样本数量。部分类别的玻璃有更多的样本（如 70 个），但是有些类别的玻璃则没有那么多样本，如有一类玻璃只有 9 个样本。

类别之间不均衡有时会导致一些问题。如果随机取样没有充分考虑样本的代表性，会导致抽样后的各类别样本所占比例与原始数据中的比例有很大的区别。为了避免这个问题，代码中采用分层抽样技术（Stratifid Sampling）。这个例子中，首先根据标签对数据进行分组（分层），然后对每组数据进行取样获得每个类别的训练、测试数据集，最后将这些针对每个类别的训练数据集组合在一起形成训练数据集，这个训练数据集的各类别样本所占的比例就与初始数据的比例一样了。

代码生成随机森林模型，然后绘制训练过程、属性的重要性排名。打印输出混淆矩阵，此矩阵显示了对于每一个类别，其样本分别有多少被预测成了其他类别。如果分类器是完美的，则在矩阵里不应该有偏离对角线的元素。

图 7-22 展示了随着决策树数目的增加，随机森林方法性能是如何改善的。随着决策树的增加，曲线通常是下降的。随着决策树的增加，改善的比率也在减少，当曲线达到最右边时，下降得已经相当缓慢了。

图 7-23 所示的条状图展示了随机森林所用属性的相对重要性排名。条状图说明有些属性对性能的贡献是相当的。这有些不同寻常。在很多情况下，属性的重要性经过前几个后会迅速下降。但在这个问题上，有几个属性具有相同的重要性。

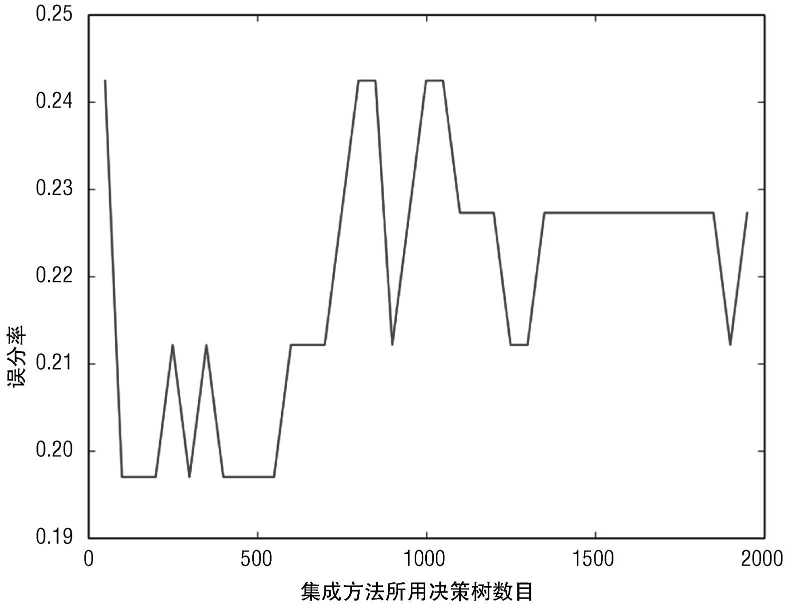


图 7-22 随机森林的整体性能

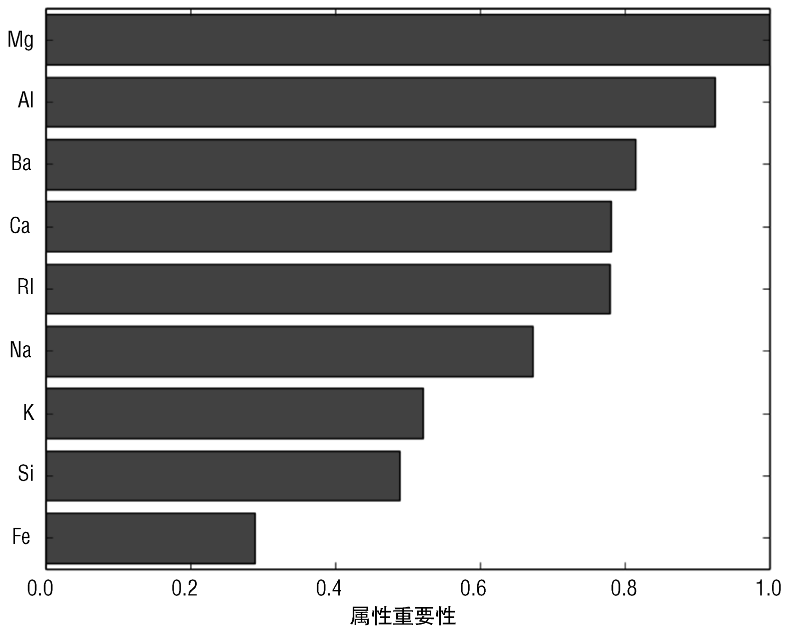


图 7-23 随机森林所用属性的相对重要性排名

7.5.3 用梯度提升法对玻璃进行分类

代码清单 7-9 除了一些小的差异外，其基本步骤与上节的使用随机森林对玻璃进行分类的过程基本一致。

代码清单 7-9 使用梯度提升法对玻璃进行分类 -glassGbm.py

```
__author__ = 'mike_bowles'

import urllib2
from math import sqrt, fabs, exp
import matplotlib.pyplot as plot
from sklearn.linear_model import enet_path
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix
from sklearn.cross_validation import train_test_split
from sklearn import ensemble
import numpy


target_url = ("https://archive.ics.uci.edu/ml/machine-learning-"
"databases/glass/glass.data")
data = urllib2.urlopen(target_url)

#arrange data into list for labels and list of lists for attributes
xList = []
for line in data:
    #split on comma
    row = line.strip().split(",")
    xList.append(row)

glassNames = numpy.array(['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca',
    'Ba', 'Fe', 'Type'])

#Separate attributes and labels
xNum = []
labels = []

for row in xList:
    labels.append(row.pop())
    l = len(row)
    #eliminate ID
    attrRow = [float(row[i]) for i in range(1, l)]
```

```
    xNum.append(attrRow)

#number of rows and columns in x matrix
nrows = len(xNum)
ncols = len(xNum[1])

#Labels are integers from 1 to 7 with no examples of 4.
#gb requires consecutive integers starting at 0
newLabels = []
labelSet = set(labels)
labelList = list(labelSet)
labelList.sort()
nlabels = len(labelList)
for l in labels:
    index = labelList.index(l)
    newLabels.append(index)

#Class populations:
#old label     new label        num of examples
#1              0                  70
#2              1                  76
#3              2                  17
#5              3                  13
#6              4                  9
#7              5                  29
#
#Drawing 30% test sample may not preserve population proportions

#stratified sampling by labels.
xTemp = [xNum[i] for i in range(nrows) if newLabels[i] == 0]
yTemp = [newLabels[i] for i in range(nrows) if newLabels[i] == 0]
xTrain, xTest, yTrain, yTest = train_test_split(xTemp, yTemp,
    test_size=0.30, random_state=531)
for iLabel in range(1, len(labelList)):
    #segregate x and y according to labels
    xTemp = [xNum[i] for i in range(nrows) if newLabels[i] == iLabel]
    yTemp = [newLabels[i] for i in range(nrows) if \
        newLabels[i] == iLabel]

    #form train and test sets on segregated subset of examples
    xTrainTemp, xTestTemp, yTrainTemp, yTestTemp = train_test_split(
        xTemp, yTemp, test_size=0.30, random_state=531)
```

```
        #accumulate
        xTrain = numpy.append(xTrain, xTrainTemp, axis=0)
        xTest = numpy.append(xTest, xTestTemp, axis=0)
        yTrain = numpy.append(yTrain, yTrainTemp, axis=0)
        yTest = numpy.append(yTest, yTestTemp, axis=0)

    #instantiate model
    nEst = 500
    depth = 3
    learnRate = 0.003
    maxFeatures = 3
    subSamp = 0.5
    glassGBMModel = ensemble.GradientBoostingClassifier(n_estimators=nEst,
                        max_depth=depth,learning_rate=learnRate,
                        max_features=maxFeatures,subsample=subSamp)
    #train
    glassGBMModel.fit(xTrain, yTrain)

    # compute auc on test set as function of ensemble size
    missClassError = []
    missClassBest = 1.0
    predictions = glassGBMModel.staged_decision_function(xTest)
    for p in predictions:
        missClass = 0
        for i in range(len(p)):
            listP = p[i].tolist()
            if listP.index(max(listP)) != yTest[i]:
                missClass += 1
        missClass = float(missClass)/len(p)

        missClassError.append(missClass)

        #capture best predictions
        if missClass < missClassBest:
            missClassBest = missClass
            pBest = p

    idxBest = missClassError.index(min(missClassError))

    #print best values
    print("Best Missclassification Error" )
```

```
print(missClassBest)
print("Number of Trees for Best Missclassification Error")
print(idxBest)

#plot training deviance and test auc's vs number of trees in ensemble
missClassError = [100*mce for mce in missClassError]
plot.figure()
plot.plot(range(1, nEst + 1), glassGBMModel.train_score_,
    label='Training Set Deviance', linestyle=":")
plot.plot(range(1, nEst + 1), missClassError, label='Test Set Error')
plot.legend(loc='upper right')
plot.xlabel('Number of Trees in Ensemble')
plot.ylabel('Deviance / Classification Error')
plot.show()

# Plot feature importance
featureImportance = glassGBMModel.feature_importances_

# normalize by max importance
featureImportance = featureImportance / featureImportance.max()

#plot variable importance
idxSorted = numpy.argsort(featureImportance)
barPos = numpy.arange(idxSorted.shape[0]) + .5
plot.barh(barPos, featureImportance[idxSorted], align='center')
plot.yticks(barPos, glassNames[idxSorted])
plot.xlabel('Variable Importance')
plot.show()

#generate confusion matrix for best prediction.
pBestList = pBest.tolist()
bestPrediction = [r.index(max(r)) for r in pBestList]
confusionMat = confusion_matrix(yTest, bestPrediction)
print('')
print("Confusion Matrix")
print(confusionMat)

# Printed Output:
#
# nEst = 500
# depth = 3
# learnRate = 0.003
```

```
# maxFeatures = None
# subSamp = 0.5
#

#
# Best Missclassification Error
# 0.242424242424
# Number of Trees for Best Missclassification Error
# 113
#
# Confusion Matrix
# [[19  1  0  0  0  1]
#  [ 3 19  0  1  0  0]
#  [ 4  1  0  0  1  0]
#  [ 0  3  0  1  0  0]
#  [ 0  0  0  0  3  0]
#  [ 0  1  0  1  0  7]]
#


# For Gradient Boosting using Random Forest base learners
# nEst = 500
# depth = 3
# learnRate = 0.003
# maxFeatures = 3
# subSamp = 0.5
#
#
#
# Best Missclassification Error
# 0.227272727273
# Number of Trees for Best Missclassification Error
# 267
#
# Confusion Matrix
# [[20  1  0  0  0  0]
#  [ 3 20  0  0  0  0]
#  [ 3  3  0  0  0  0]
#  [ 0  4  0  0  0  0]
#  [ 0  0  0  0  3  0]
#  [ 0  2  0  0  0  7]]
```

如前所述，在 GradientBoostingClassifier 类中可以使用迭代版本的梯度提升法，这样在梯度提升的训练过程中每一步都可以生成预测结果。

7.5.4 评估在梯度提升法中使用随机森林基学习器的好处

在代码的最后可以看到 max_features 分别设为 None 和 20 时，梯度提升法的预测结果。第一个取值是指训练普通的决策树，就像最初提出梯度提升法算法的论文建议的那样。第二个取值是指将随机森林作为基学习器，在决策树的每个节点进行数据分割时不考虑所有的属性，只随机考虑 max_features 个属性。这实际上是梯度提升和随机森林两种方法的混合。

训练数据集的偏差和测试数据集上的误分类误差如图 7-24 所示。偏差展示了训练的过程。测试数据集的误分类误差用来确定模型是否过拟合。算法没有过拟合，但是在决策树数目达到 200 左右后，性能也不再提升，因此可以终止测试。

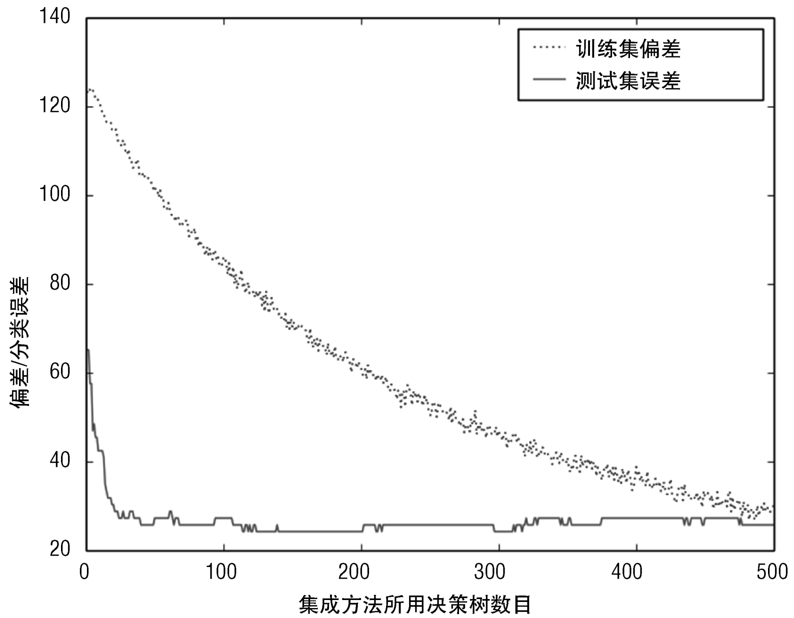


图 7-24 用梯度提升法分类玻璃的性能

图 7-25 为梯度提升法中各属性的重要性排名。此图不太寻常，有相当多属性的重要性差不多。通常是有几个属性非常重要，然后后续属性的重要性迅速下降。

当 max_features 取值为 20 时，其训练数据偏差和测试数据误分类误差如图 7-26 所示，这时使用的是随机森林基学习器。这导致误分类率改善了大概 10%。但在图 7-26 中察觉

不到，这种轻微的改善并不能改变图 7-26 中曲线的基本特征。

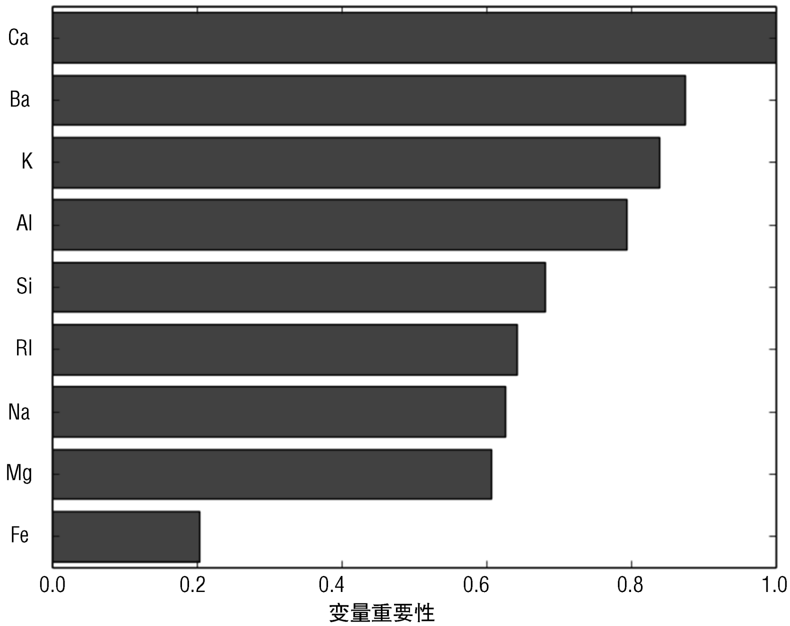


图 7-25　用梯度提升对玻璃进行分类：各属性重要性排名

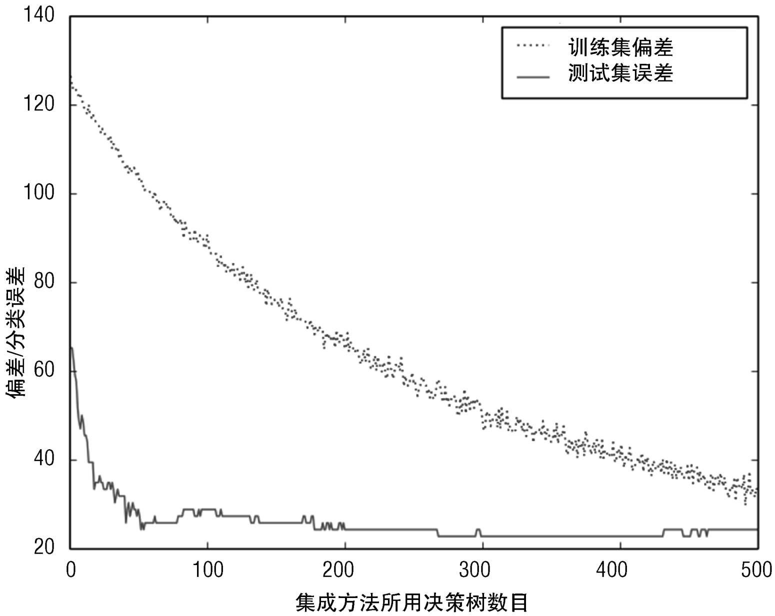


图 7-26　基于随机森林基学习器的梯度提升法对玻璃进行分类：性能与决策树数目的关系

基于随机森林基学习器的梯度提升法所用属性的重要性排名如图 7-27 所示。此图各属性的排名与图 7-25 中的有些不同。有些属性都出现在两个排名的前五，但是有些属性在一个排名中出现在前五，但在另一个排名中却出现在最后。这两个图都显示了某些属性具有类似的重要，这可能是这两个排名有些不稳定的原因。

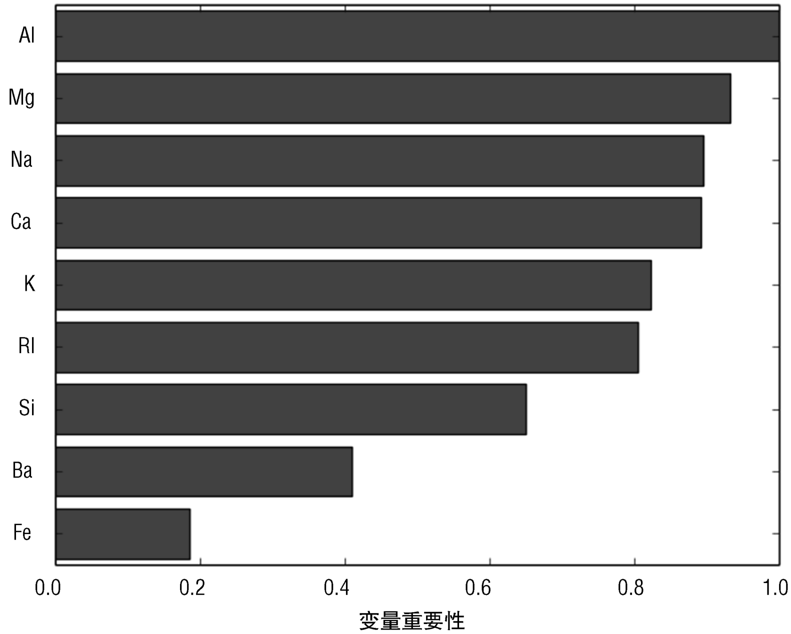


图 7-27　基于随机森林基学习器的梯度提升法对玻璃进行分类：各属性重要性排名

7.6　算法比较

本章出现的算法在时间和性能上的对比如表 7-1 所示。时间是完成一轮完整训练所需的时间。训练随机森林的部分代码训练了一系列不同规模的模型。在这种情况下，最长训练时间作为所需时间。“算法”列的其他用来展示训练行为随决策树数目的变化情况。相似地，对于惩罚线性回归算法，多数情况使用 10 折交叉验证，但也有的只使用了一个测试数据集。单个测试数据集只需要一轮训练时间，然而对于 10 折交叉验证需要进行 10 轮训练。对于引入 10 折交叉验证的算法，表 7-1 中只显示了 10 轮时间的十分之一。

除了玻璃分类数据集（多类别分类问题），惩罚线性回归的训练时间要比梯度提升和随机森林快一个数量级。通常随机森林和梯度提升的性能要优于惩罚线性回归。惩罚线性

回归只在某些数据集上性能接近随机森林和梯度提升。若要在红酒数据集上达到接近的性能，需要进行基扩展（basis expansion）。基扩展如果应用到其他数据集上，也可能会带来性能的进一步提升。

表 7-1 性能与训练时间的对比

数据集	算法	训练时间	性能	性能评价标准
玻璃	随机森林 2000 决策树	2.354401	0.227272727273	分类错误
玻璃	梯度提升法 500 决策树	3.879308	0.227272727273	分类错误
玻璃	套索（lasso）回归	12.296948	0.373831775701	分类错误
岩石与水雷	随机森林 2000 决策树	2.760755	0.950304259635	曲线下面积
岩石与水雷	梯度提升法 2000 决策树	4.201122	0.956389452333	曲线下面积
岩石与水雷	ElasticNet 回归	0.519870*	0.868672796508	曲线下面积
鲍鱼	随机森林 500 决策树	8.060850	4.30971555911	均方误差
鲍鱼	梯度提升法 2000 决策树	22.726849	4.22969363284	均方误差
红酒	随机森林 500 决策树	2.665874	0.314125711509	均方误差
红酒	梯度提升法 2000 决策树	13.081342	0.313361215728	均方误差
红酒	套索扩展回归	0.646788*	0.434528740430	均方误差

* 标星的时间是单独一次交叉验证的时间。这些方法根据 n 折交叉验证技术会重复训练几次。但是其他方法只在一个测试数据集上进行测试。所以采用单独一次交叉验证的时间与其他算法对比。

随机森林和梯度提升法两者的性能很接近。只不过某些时候，一个算法要比另一个算法需要训练更多的决策树才能达到相似的性能。随机森林和梯度提升法的训练时间大致也一样。在两者训练时间不一致时，一个算法的训练时间可能远远超出了它必需的时间。例如，针对鲍鱼数据集，梯度提升法的测试数据误差在 1 000 步（决策树）时就已经很平坦了，但是训练还是一直持续到 2 000。如果进行修正就可以砍掉梯度提升一半的训练时间，就会使两者的训练时间保持一致。上述情况对红酒数据集也同样成立。

小结

本章介绍了 Python 实现的集成方法工具包。实例代码展示了使用这些方法针对不同类型的问题构建模型。本章涵盖了回归问题、二分类问题、多类别分类问题，并讨论了一些变化，如何类别属性的编码、分层取样等。这些例子涵盖了可能在实践中遇到的不同问题类型。

这里的例子也展示了集成方法的重要特征：为什么对于数据科学集成方法是首选的原因。集成方法相对易于使用。它们不需要调很多参数。它们可以给出属性的重要性信息，有利于模型开发早期阶段的对比和分析，集成方法通常也可以获得最佳的性能。

本章介绍了相关 Python 工具包的使用。第 6 章的背景知识帮助理解这些参数的设置及其调整。通过观察实例代码中参数的设置，可以帮助尝试使用这些软件包。

在本章的最后对比分析各种算法。集成方法通常可以获得最佳的性能。惩罚回归方法通常比集成方法快，在某些情况下，可以获得接近的性能。

参考文献

1. sklearn documentation for RandomForestRegressor, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

2. Leo Breiman. (2001). “Random Forests.” Machine Learning , 45 (1): 5–32.doi:10.1023/A:10109334043243. J. H. Friedman. “Greedy Function Approximation: A Gradient BoostingMachine,” https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

3. J. H. Friedman. “Greedy Function Approximation: A Gradient Boosting Machine,” https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

4. sklearn documentation for RandomForestRegressor, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.

RandomForestRegressor.html

5. L. Breiman, “Bagging predictors,” http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf

6. Tin Ho. (1998). “The Random Subspace Method for Constructing DecisionForests.” IEEE Transactions on Pattern Analysis and Machine Intelligence ,20 (8): 832–844. doi:10.1109/34.709601

7. J. H. Friedman. “Greedy Function Approximation: A Gradient BoostingMachine,” https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

8. J. H. Friedman. “Stochastic Gradient Boosting,” https://statweb.stanford.edu/~jhf/ftp/

stobst.pdf

9. sklearn documentation for GradientBoostingRegressor, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

10. J. H. Friedman. "Greedy Function Approximation: A Gradient BoostingMachine," https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

11. J. H. Friedman. "Stochastic Gradient Boosting," https://statweb.stanford.edu/~jhf/ftp/stobst.pdf

12. J. H. Friedman. "Stochastic Gradient Boosting," https://statweb.stanford.edu/~jhf/ftp/stobst.pdf

13. sklearn documentation for RandomForestClassifier, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

14. sklearn documentation for GradientBoostingClassifier, http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

欢迎来到异步社区！

异步社区的来历

异步社区（www.epubit.com.cn）是人民邮电出版社旗下 IT 专业图书旗舰社区，于 2015 年 8 月上线运营。

异步社区依托于人民邮电出版社 20 余年的 IT 专业优质出版资源和编辑策划团队，打造传统出版与电子出版和自出版结合、纸质书与电子书结合、传统印刷与 POD 按需印刷结合的出版平台，提供最新技术资讯，为作者和读者打造交流互动的平台。



社区里都有什么?

购买图书

我们出版的图书涵盖主流 IT 技术，在编程语言、Web 技术、数据科学等领域有众多经典畅销图书。社区现已上线图书 1000 余种，电子书 400 多种，部分新书实现纸书、电子书同步出版。我们还会定期发布新书书讯。

下载资源

社区内提供随书附赠的资源，如书中的案例或程序源代码。

另外，社区还提供了大量的免费电子书，只要注册成为社区用户就可以免费下载。

与作译者互动

很多图书的作译者已经入驻社区，您可以关注他们，咨询技术问题；可以阅读不断更新的技术文章，听作译者和编辑畅聊好书背后有趣的故事；还可以参与社区的作者访谈栏目，向您关注的作者提出采访题目。

灵活优惠的购书

您可以方便地下单购买纸质图书或电子图书，纸质图书直接从人民邮电出版社书库发货，电子书提供多种阅读格式。

对于重磅新书，社区提供预售和新书首发服务，用户可以第一时间买到心仪的新书。

用户帐户中的积分可以用于购书优惠。100 积分 =1 元，购买图书时，在 [0] 使用积分 里填入可使用的积分数值，即可扣减相应金额。

Python 机器学习

预测分析核心算法

用 Python 分析数据、预测结果的简单高效的方式

机器学习关注于预测，其核心是一种基于数学和算法的技术，要掌握该技术，需要对数学及统计概念有深入理解，能够熟练使用 R 语言或者其他编程语言。

本书通过集中介绍两类可以进行有效预测的机器学习算法，展示了如何使用 Python 编程语言完成机器学习任务，从而降低机器学习难度，使机器学习能够被更广泛的人群掌握。

作者利用多年的机器学习经验带领读者设计、构建并实现自己的机器学习方案。本书尽可能地用简单的术语来介绍算法，避免复杂的数学推导，同时提供了示例代码帮助读者迅速上手。读者会很快深入了解模型构建背后的原理，不论简单问题还是复杂问题，读者都可以学会如何找到问题的最佳解决算法。书中详细的示例，给出了具体的可修改的代码，展示了机器学习机理，涵盖了线性回归和集成方法，帮助理解使用机器学习方法的基本流程。

本书为不具备数学或统计背景的读者量身打造，详细介绍了如何：

- 针对任务选择合适算法；
- 学习数据处理机制，准备数据；
- 掌握 Python 机器学习核心算法包；
- 构建实用的多功能预测模型。
- 对不同目的的应用训练好的模型；
- 评估模型性能以保证应用效果；
- 使用示例代码设计和构建你自己的模型；

作者简介：

Michael Bowles 在硅谷黑客道场教授机器学习，提供机器学习项目咨询，同时参与了多家创业公司，涉及的领域包括生物信息学、金融高频交易等。他在麻省理工学院获得助理教授教职后，创建并运营了两家硅谷创业公司，这两家公司都已成功上市。他在黑客道场的课程往往听者云集并且好评颇多。

异步社区 人民邮电出版社 www.epubit.com.cn

异步社区 www.epubit.com.cn
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-43373-2

9 787115 433732 >

封面设计：董志桢

分类建议：计算机／机器学习
人民邮电出版社网址：www.ptpress.com.cn